Exact Apparent Motion via Eulerian Flow Maps: Real-Time Animated Streamlines for Immersive Exploration

Category: n/a

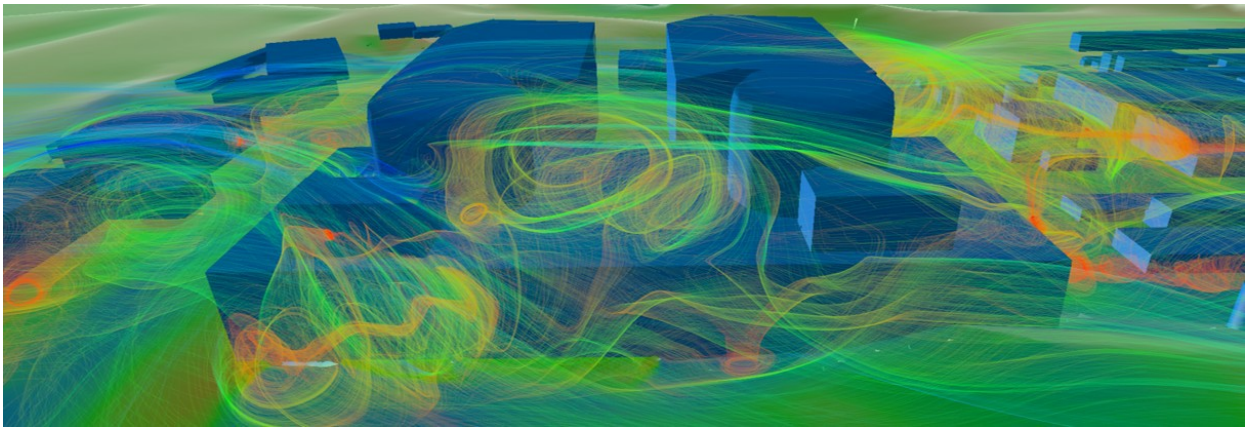


Fig. 1: Animated streamline visualization of airflow around a power plant in an urban setting. Composed of millions of moving line segments, the rendering captures continuous flow trajectories and demonstrates intricate fluid dynamics and transport mechanisms within the built environment.

Abstract—Streamlines are a fundamental representation for flow visualization, enabling analysis of internal flow structures, transport phenomena, and vortical behavior in complex vector fields. Extending this technique into extended reality (XR) environments introduces new opportunities for situated, embodied exploration of large-scale computational fluid dynamics (CFD) datasets. However, existing approaches predominantly address seed placement and occlusion management, while real-time rendering of dense animated streamlines and physically consistent motion encoding remain underexplored, especially in immersive settings where optical flow sensitivity and spatial presence mutually reinforce analytic interpretation. To address these challenges, we propose a novel real-time animated streamline approach integrated into an interactive XR platform, enabling the progressive and insight-driven exploration of large-scale flow datasets. At its core, the method introduces an Eulerian Flow Map algorithm coupled with spacetime parallelism, leveraging pixel-wise temporal phase shifts for motion alignment to seamlessly capture the continuous evolution of streamlets, supported by rigorous proofs. Accelerated by GPU computing, our approach enables rendering hundreds of millions of dynamic streamlets, achieving an exact apparent motion that faithfully represents the underlying physical flow velocity and direction. Furthermore, the method supports versatile motion patterns and advanced rendering features, tunable via multi-parameter controls to facilitate uncertainty perception. Ultimately, we provide diverse engineering implementations of our method, contributing to the establishment of a powerful and intuitive framework for deciphering complex flow behaviors and advancing insights into fluid fields. A free copy of this paper and all supplemental materials are available at https://reshyx.github.io/Dense-Animated-Streamlines.

Index Terms—Animated streamline, GPU, extended reality, interactive, vector field visualization, motion visualization

1 Introduction

Flow simulation is widely used in aerospace, environmental science, energy systems, and biomedicine. Modern computational fluid dynamics (CFD) methods can generate high-resolution, time-resolved datasets that capture detailed flow structures [8]. While these data improve physical accuracy, they also create significant challenges for visualization, especially when users need to explore large-scale, three-dimensional flow fields interactively. Furthermore, the use of diverse visualization and interaction techniques, whether geometric or image-based, automated or interactive, often leads to varied perceptual outcomes. Both the large-scale datasets and the diversity of approaches highlight the need to extract salient features amidst uncertainty to support focused and effective interpretation [31].

Extended Reality (XR) has long been recognized for its potential in scientific visualization, providing an intuitive perspective that allows researchers to visualize and interact directly with massive 3D models [38]. By making abstract scientific structures (such as magnetic and velocity fields) more tangible, XR supports situated visualization and embodied exploration, presenting rich visual information with reduced cognitive overload. Significant progress has been made in the immersive visualization of flow fields, such as visualizing particle trajectories and vector glyphs within virtual reality (VR) [49,50] and investigating VR's potential in the CFD field for rapidly constructing mental models that enhance flow behavior comprehension [17]. More recently, research has brought these approaches closer to practical usability [43,60], notably by integrating XR into ParaView. There has also been active innovation in exploring immersive paradigms for fluid visualization, optimizing the user experience by integrating multi-sensory feedback mechanisms, including haptic, olfactory, and vibrational elements [24,44,62]. Concurrently, some offline methods have been applied to practical CFD education in deployed CFD graduate-level curricula [3], and real-time, fast Position-Based Fluid (PBF) methods have been successfully implemented for fluid visual effects [6,7,32]. User studies indicate a preference for these interactable physical simulations [69], despite their reduced physical accuracy compared to rigorous CFD methods.

However, effectively exploring large-scale, high-fidelity CFD datasets interactively within XR remains a significant technical hurdle. To navigate these complex datasets, streamlines are particularly valued for their physical intuitiveness. They illustrate the trajectories of fluid particles and clearly reveal key flow structures such as separation zones, vortex cores, and recirculation regions. Their topological co-

herence makes them especially suitable for interpreting internal flow phenomena, including separation line attachment effects and energy dissipation within vortices. Therefore, we utilize interactive streamline generation as a primary tool to explore large-scale flow fields. Prior work on static streamlines is extensive: Uncertainty and error are highly sensitive to numerical integration and interpolation choices [31]; seed placement strongly shapes the geometric depiction of structures such as vortices [34]; and rendering decisions, including occlusion management, lighting, and transparency for dense primitives, further contribute to variability and uncertainty in the resulting imagery [15,37]. Although the effectiveness of animated streamlines for flow pattern visualization has been validated [70], such efforts have been confined to 2D and sparse streamlets. Dynamic 3D flow visualization remains underexplored, especially in XR, where spatial presence and sensitivity to optical flow are mutually reinforcing [12, 46, 52]. This gap reflects, in part, limited real-time rendering and interaction in conventional analysis workflows. Generating and rendering large-scale dynamic flow fields thus remains difficult, and hardware constraints together with inefficient strategies still impede immersion and embodiment in scientific visualization.

For motion visualization of flow fields, beyond computational efficiency, existing research often neglects physical fidelity, which can introduce artifacts, bias perceived speed, and undermine analytic reliability. Motivated by these limitations, we target a dynamic, streamline-based framework for real-time XR presentation that scales to large, complex CFD datasets while conveying visually faithful cues. In this paper, we propose an efficient, streamline-based method grounded in Eulerian motion mapping. By prioritizing physical consistency, it aligns the apparent motion of streamlets with the underlying velocity field and avoids integration-induced artifacts. Researchers can explore flow structures immersively, including ubiquitous vortex phenomena (see Fig. 2), and gain a more complete understanding of the flow in two respects: **1)** exploring fine unstructured grids in real-time by interactively seeding dense streamlines in regions of interest; and **2)** representing fluid motion and structure through animation and the geometric properties of streamlines, governed by multiple tunable parameters.

Our key contributions are summarized as follows:

- **A physically consistent animation model for streamlines**: We introduce a time-shift-based formulation that ensures exact apparent motion, where the perceived motion of streamlines strictly follows the underlying integration time.
- **An Eulerian flow map for animation**: We reformulate streamline animation as a spatiotemporal phase mapping, enabling motion to be computed without explicit particle advection and decoupling spatial and temporal components.
- **A scalable, multi-level CPU-GPU parallel pipeline**: Our approach supports instant interaction through parallel streamline processing, and enables real-time rendering of hundreds of millions of animated primitives via independent pixel-wise shading.
- **An interactive XR system for immersive flow exploration**: We integrate the method into an extended reality environment, supporting progressive seeding, parameter control, and intuitive exploration of complex flow structures.

2 Related Work

Immersive-SciVisualization: XR has long been recognized as a powerful tool for scientific visualization [57], and continues to prove effective today [23, 33]. By providing an immersive perspective, XR enables users to interact directly with complex 3D data, making abstract scientific structures more accessible. For instance, XR has been used to visualize organs for disease diagnosis [55, 76] and provide cellular-level insights [68], which are difficult to achieve in reality. It also supports design optimization by using visual cues such as color to highlight intricate structures and encode physical properties like pressure and velocity [56, 77]. Beyond biomedicine, XR effectively conveys abstract scientific concepts, such as astronomical dynamics [36], molecular structures [42], and vector fields including magnetic and velocity data [28]. Moreover, XR has been shown to handle extremely large datasets while still providing intuitive interaction [10]. These applications highlight XR's value in both enhancing geometric understanding and making abstract or large-scale data more tangible.

The Visualization Toolkit (VTK) [51] has become a cornerstone for 3D graphics, image processing, and scientific visualization. Building on VTK, ParaView [1] was later developed as a scalable open-source system, and by 2016 it included VR plugins for immersive visualization. Research on immersive visualization with VTK predates these developments [22], and in 2017 Kitware outlined four key strategies for integrating VTK with immersive systems like CAVE and head-mounted displays (HMDs) [43]. These strategies include geometry transport [26, 58], OpenGL context sharing [75], embedding VR toolkits [66], and OpenGL intercept. Our implementation of some of these methods, particularly the OpenXR plugin of ParaView and the C# wrapper of VTK - "Activiz" for Unity, revealed their limitations for our intended goals. We also examined collaborative frameworks such as NVIDIA Omniverse. A more detailed discussion of this evaluation is provided in the supplementary material.

Streamlines-Uncertainty: Streamline visualization is subject to uncertainties from multiple sources. Prior research has largely focused on optimizing seed point placement [48] and improving rendering and coloring techniques [20]. However, handling large-scale flow fields imposes heavy computational demands on automated seed placement. To balance flexibility and efficiency, we adopt interactive seeding in user-defined regions of interest, allowing targeted exploration of flow fields. In dense streamline rendering, visual clutter and poor rendering quality can obscure meaningful structures; advanced lighting and shadowing methods are important to preserve spatial detail [11, 37]. Transparency and shadow-based effects have also been combined with animation to capture uncertainty in flow visualization [4].

Although most of these techniques are not directly applicable to real-time rendering or general datasets, motivated by these studies, we implemented multiple shading and rendering techniques that users can adjust to accommodate diverse scenarios and analysis requirements. We also proposed various parameterized controls for the animation effects, extending uncertainty motion perception for further investigation, inspired by user research on animation effects in 2D flow fields [70].

Animation-Visualization: Motion visualization enhances human perception of flow fields, and animated streamlines have been shown to be particularly effective [47, 70]. For example, Ding et al. [9] studied how encoding choices, such as velocity, particle length, density, and color saturation, affect motion perception in 2D settings. Our work extends this direction by introducing multiple animation representations, controlled through parametric functions, to generate diverse motion patterns for immersive flow analysis. Existing animation techniques generally fall into two categories. The first, *Advection & Blend*, includes methods such as OLIC [72], IBFV [67], surface-based approaches [59], and UFLIC [53], as well as texture-based and feature-based blending techniques [2, 30, 74]. Some extensions into 3D voxels [61] and real-time rendering [73] exist, but their scalability to large 3D flow datasets is limited. The second category, *Precompute & Offset*, has roots in interactive streamline work by Van et al. [65] and Jobard et al. [18, 19], later expanded in FROLIC [71] and real-time fragment-based encoding [25]. Recent efforts include 2D wind field studies [80] and wave simulation using repeated asymmetric patterns [78].

Despite these advances, efficient visualization of dense 3D animated streamlines remains largely unexplored, both in implementations and in principle. This gap has hindered the wider adoption of animated streamline visualization for large-scale flow fields. For example, ParaView's built-in *Advection & Blend* animations still face significant limitations (Sec. 10). Although some methodologies consider the mapping relationships of varying velocities, they do not reveal the underlying principles, leading to limitations when handling adaptive time steps in streamline integration. Overall, challenges in large-scale 3D flow visualization with physical fidelity, dense streamline rendering, and real-time immersive performance remain open problems, and form the focus of our work.

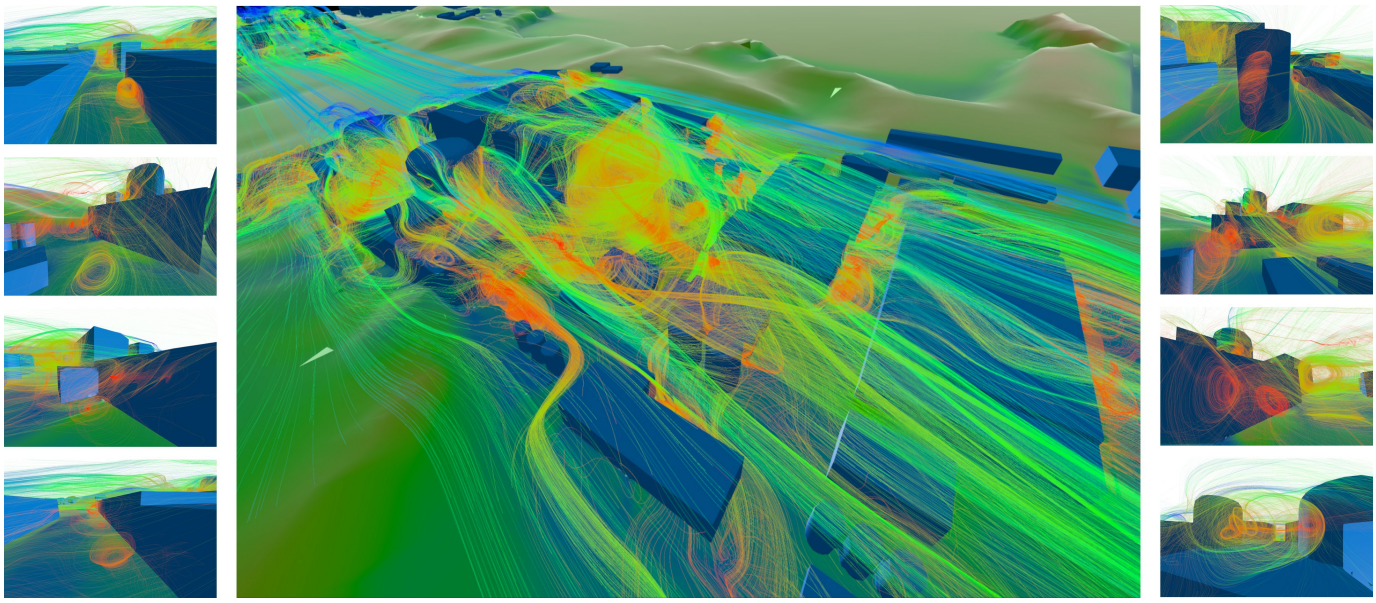


Fig. 2: Ubiquitous vortex: An example of the streamline visualization in a bay area, including both global view and several localized detailed views of the vortices. High-resolution image; enlargement is recommended to highlight details and improve visual immersion.

3 Method Overview

Our approach consists of two stages: (1) parallel streamline generation and (2) GPU-based animation using an Eulerian formulation. The key idea is to decouple spatial geometry from temporal evolution, enabling efficient and independent motion computation during rendering.

In the *streamline generation* stage, we compute streamlines with both accuracy and efficiency in mind. Accurate integration ensures geometric fidelity and provides the temporal information required for physically consistent animation. We accelerate this process using spatial partitioning, thread-level parallelization, and bidirectional integration from seed points. The resulting segments are connected via a prefix-sum-based reconstruction, improving data compactness and continuity. The final data, including vertex positions, topology, and attributes, are stored in GPU-accessible buffers.

In the *streamline animation* phase, we adopt a GPU-parallel approach that decouples spatial and temporal computations. Instead of explicit particle advection, we model animation as a *spatiotemporal phase shift* applied to precomputed streamlines. This mapping is derived from integration time, allowing each vertex to evolve according to its own local temporal parameter.

We implement this formulation using a spacetime shear transformation that maps spatial progression into the temporal domain. By explicitly incorporating integration time, our method ensures that apparent motion remains consistent with the underlying flow. The fully parallel per-vertex (or fragment) computation enables scalable, real-time rendering of dense 3D flow fields.

4 Streamline Integration

The precision of numerical integration methods primarily introduces geometric visualization errors [31]. For animation-based approaches, however, we argue that the time-step mapping of the integration scheme is fundamental to the correctness of the perceived motion. To establish an appropriate kinematic representation, we first distinguish the integration step (pseudo-time step) from the animation time step. A streamline is mathematically defined by the alignment of its tangent vector with the underlying vector field $\mathbf{V}(\mathbf{x})$ at each spatial point. This can be expressed by the vanishing cross product: $\mathrm{d}\mathbf{r} \times \mathbf{V}(\mathbf{x}) = \mathbf{0}$. This relation implies the existence of a scalar function $\lambda(\mathbf{x})$ such that

$$\mathrm{d}\mathbf{r} = \lambda(\mathbf{x})\mathbf{V}(\mathbf{x}), \tag{1}$$

which establishes a proportional relationship between the infinitesimal displacement and the vector field. To calculate streamlines, there is a pseudo-temporal variable s that uniquely identifies positions along a trajectory. While λ as an infinitesimal step $\mathrm{d}s$ ($\lambda(x) \to 0$), the streamline equation becomes a velocity-driven autonomous system:

$$\begin{cases} \mathrm{d}\mathbf{r} = \mathbf{V}(\mathbf{x})\,\mathrm{d}s \\ s(\mathbf{x}) = \int \lambda(\mathbf{x})\,d\mathbf{x} + C \end{cases} \quad \Rightarrow \quad \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}s} = \mathbf{V}\big(\mathbf{x}(s)\big), \tag{2}$$

where $\mathbf{r}(s) \equiv \mathbf{x}(s)$ represents the corresponding position at the same. For three-dimensional vector fields with initial condition (seed point) $\mathbf{x}_0 = [x_0, y_0, z_0]^\mathsf{T}$ at $s = s_0$, this formulation is reduced to the following integral equation:

$$\mathbf{x}(s) = \mathbf{x}_0 + \int_{s_0}^{s} \mathbf{V}\big(\mathbf{x}(\tau)\big)\,\mathrm{d}\tau. \tag{3}$$

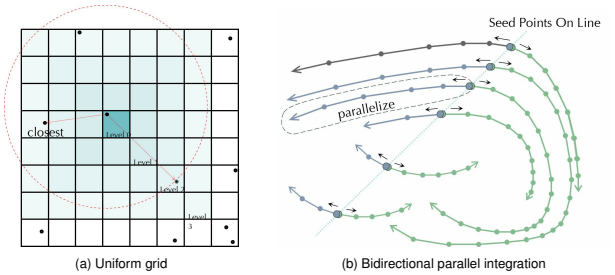
In practice, streamlines are computed by numerically solving this integral equation, which employs discretization schemes to approximate the integral over incremental intervals $[s_n, s_{n+1}]$:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \int_{s_n}^{s_{n+1}} \mathbf{V}(\mathbf{x}(\tau))\,\mathrm{d}\tau = \mathbf{x}_n + \mathbf{V}\big(\mathbf{x}(s')\big)\Delta s, \tag{4}$$

where $\Delta s = s_{n+1} - s_n$ and $s' \in [s_n, s_{n+1}]$ denotes an intermediate sampling point. Streamlines are thus parameterized by the pseudo-time step Δs and advanced using explicit integration schemes. To improve accuracy, we use multiple intermediate velocity evaluations with extrapolation, achieving higher-order integration. An adaptive step-size controller dynamically adjusts Δs based on local error estimates, balancing accuracy and computational efficiency. In our implementation, we utilize fixed-step RK4 [14] and adaptive step-size RK45 (Cash–Karp) [5] methods to evaluate and analyze the velocity mapping accuracy.

To reduce the latency of interactive streamline generation for large-scale data, a key challenge for our pipeline is to generate large numbers of streamlines efficiently. We address this by using optimized spatial data structures for nearest-neighbor search during velocity interpolation. Our tests indicate that while spatial partitioning methods for sparse data (such as KD-trees, Octrees, or CellTrees [13]) perform well under specific scenarios, a uniform grid bucketing approach (Fig. 3a) is generally faster for static datasets leveraging massive parallelism. For streamline generation, at each integration step, the interpolation requires locating the nearest data point to a query position p_0. This is achieved through a multi-level incremental grid search that leverages spatial locality for

cache-efficient traversal: first, the Chebyshev distance (L_∞) defines the expansion of search layers to identify an initial candidate p_1. Subsequently, a refinement is performed within the Euclidean radius (L_2) $R = ||p_0 - p_1||_2$. To account for the discrepancy between the square search layers and the spherical search radius, all cells intersecting the Euclidean sphere are examined to ensure the global nearest neighbor is captured.



(a) Uniform grid (b) Bidirectional parallel integration

Fig. 3: Space partitioning and parallel integration.

Another strategy for accelerating streamline computation is the use of parallel methods. A commonly employed approach is particle-level parallelism, where seed points are distributed across computational threads or processes to enable simultaneous integration. This method requires careful attention to load balancing, especially during streamline propagation, to ensure uniform computational efficiency [79]. Alternatively, data-level parallelism decomposes the spatial domain, using out-of-core techniques [63] or distributed computing frameworks [41]. In this approach, streamline integration is performed independently within local subdomains and followed by synchronization across subdomain boundaries to maintain global consistency. Parallelization across algorithmic stages, such as parallel-in-stages extrapolation, has been explored but often yields limited speedup due to inter-stage dependencies [21,64]. Meanwhile, parallel-in-time methods, such as the Parareal algorithm [27], enable implicit solvers to achieve spacetime parallelism and support larger time steps. However, these methods are generally not suitable for the portable and visually smooth generation of streamlines required in real-time, immersive visualization environments.

In our particle-level parallelism implementation, we employ bidirectional integration to improve computational efficiency (Fig. 3b). However, variability in integration runtimes, caused by differing termination conditions such as domain exit, stagnation, or maximum step limits, creates challenges for memory layout and data distribution, as previously discussed by Lawrence et al. [39]. The resulting irregular and unpredictable streamline lengths make uniform pre-allocation of memory impractical. To address these challenges, as shown in Fig. 4, the forward-integrated streamline is first reversed by swapping vertex data in memory, reducing dependencies and enabling instruction-level parallelism in subsequent steps. We then apply a prefix-sum computation to the merged streamline pairs to determine their final memory addresses. Finally, all streamlines are copied in parallel into a compact memory block within preallocated address space, achieving efficient and scalable memory management.

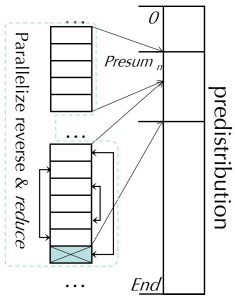


Fig. 4: Mem. Reordering

5 Oriented Streamlets on Lines

To animate streamlines, we adopt a streamlet-based approach inspired by Color Table Animation and Moving Texture [35,54], where each streamline is divided into successive, visually coherent components (streamlets), producing a trailing effect that enhances the perception of flow direction and trajectory, even in static visualizations [47,70,72]. Such fading or repeating patterns can be represented using kernel functions, as in the FROLIC framework. A simple example is the modulo (MOD) function, which creates periodic droplet-like segments via convolution. Conceptually, a streamlet can be regarded as a snapshot of the same entity at successive pseudo-time steps, together encoding the temporal evolution of motion along the streamline. This idea is closely related to motion maps, originally introduced by Jobard et al. [18]. To implement the approach, strategies for handling motion mapping with velocity variations are required based on Lefer and Jobard's method [25]. They constructed a texture map to assign indices $Index_{P_i}$ for each pixel in 2D:

$$\begin{cases} S_{P_0} = random(N), \\ S_{P_i} = S_{P_{i-1}} + k(V_{P_i}), \\ Index_{P_i} = [S_{P_i}] \ mod \ N. \end{cases} \tag{5}$$

Here, $k(V_{P_i})$ denotes a function that determines the incremental step S as the streamline progresses from P_{i-1} to P_i, based on the local velocity V_{P_i}. By modeling $k(v)$ as a linear velocity-dependent mapping $k(v) = a \times v + b$ and imposing boundary conditions to constrain the mapped length $L \in [L_{\min}, L_{\max}]$, we have

$$k(v) = \frac{N}{L_{\max}} \left(\frac{v - V_{\min}}{V_{\max} - V_{\min}} \right) + \frac{N V_{\max}}{L_{\max} V_{\min}} \left(\frac{V_{\max} - v}{V_{\max} - V_{\min}} \right). \tag{6}$$

However, this formulation faces two flaws in our research: (1) pixel-based encoding is unsuitable for 3D environments with occlusion, while voxel-based methods like 3D IBFV remain prohibitively expensive; (2) the linear velocity mapping $k(v)$ assumes fixed-step integration and completely breaks down in adaptive-step scenarios—a failure that directly undermines the fidelity of the resulting apparent motion. To address these shortcomings, we propose integration-aware encoding for vertex-based polylines. This approach also reflects the physical intuition that animation time intervals should be strictly synchronized with numerical integration time steps; specifically, the visual progression along a streamline should reflect the actual elapsed physical time, rather than relying on an ad-hoc, "velocity-aware" mapping. Instead of per-pixel assignment, we encode the vertices of preprocessed and sorted streamlines. We introduce a normalized scalar variable x along each streamline and apply a scaling factor β to ensure that each streamlet maintains a fixed period k. (β/k can always be reduced to a single scaling coefficient, we distinguish them here to assign them distinct physical meanings.) The mapping function $\mathscr{M}$ is then defined as:

$$\mathscr{M} : \mathbb{N} \to \mathbb{R}_0^+ \, | \, x = \frac{s(\mathbf{x}_i)}{k} \beta \tag{7}$$

Figure 5 illustrates our approach for generating animated effects on streamlines by focusing on a single streamline within the lineset. Each vertex is assigned a distinct color, which effectively segments the streamline into streamlets. Transparency along the streamline varies according to a sawtooth wave pattern, enhancing the perception of motion and directionality. To further refine the visual encoding for trajectory trailing effects, we introduce a composite mapping:

$$\phi(x) = f \circ g(x), \quad f(x) = \alpha(1 - \tau x)^\lambda \tag{8}$$

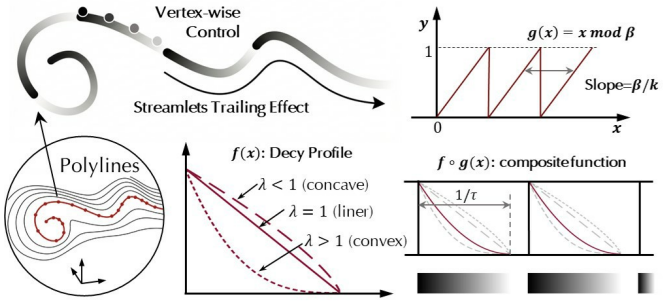


Fig. 5: Parametric flow mapping for streamlines in 3D.

where $g(x) = x \bmod \beta$ serves as a periodic windowing operator. It maps the global streamline coordinate x into a localized interval $[0,1)$, effectively sampling a specific segment of the decay profile $f(x)$ to be tiled cyclically. The function parameters provide fine-grained control over the animation's global distribution and local geometry:

- α (Scaling Factor): Controls the overall magnitude of the visual encoding (e.g., color or opacity), acting as a master gain for the animation's prominence.
- β/k (Segmentation Length): Dictates the spatial partitioning of streamlets along the streamline; it determines the global frequency of the animated segments.
- τ (streamlets' Truncation): Adjusts the visible length of the trailing effect within the window, effectively cutting off the tail.
- λ (Decay Curvature): Governs the transparency attenuation rate, enabling diverse decay profiles such as linear ($\lambda = 1$), convex, or concave shapes.

By tuning these parameters, we can achieve diverse trailing aesthetics, significantly enhancing the perception of complex flow structures.

6 Advection via Time Shift

Time-shift methods represents the theoretical culmination of these animation paradigms. While these techniques are effective at generating animated visual effects, the intrinsic mathematical relationship between the shear transformations and physical motion has remained largely unexamined. Can a shear transformation of arbitrary intensity accurately presented physical advection? Spacetime method [40] provides a unified four-dimensional spacetime framework in which spatial and temporal dimensions are interwoven and all events coexist. This conceptualization motivates our computational reinterpretation of advection dynamics as a shearing process in higher-dimensional space.

We formulate advection as a shear deformation in spacetime, where the convection equation determines the shear intensity, γ. As illustrated in Fig. 6a, extruding the static profile $\phi(x)$ (Eq. (8)) along the temporal axis (t) constructs a 3D spacetime surface. A cross-section of this surface at any constant t represents the function's instantaneous state. For a straight extrusion, advancing the t-plane yields an invariant profile, indicating zero spatial displacement. However, applying a temporal shear transformation (Fig. 6b) slants this surface. Consequently, as the t-plane progresses, the intersecting profile continuously translates along the spatial axis. Ultimately, this mechanism translates temporal progression directly into perceived spatial motion.

Building on the preceding discussion, we extend motion visualization by embedding the streamline decay profile function in a higher-dimensional framework. This approach emphasizes localized, pointwise computations rather than global advection-based transformations. To formalize this concept, we extend the formulation in Eq. (9) by introducing the sets T and X, where $t \in T$ represents time and $x \in X$ denotes positions along the curve. We define a new mapping $\mathscr{T} : X \times T \to \mathbb{R}$, as illustrated in Fig. 6a, allowing us to derive a time-invariant function:

$$\mathscr{T}(x,t) = \mathscr{T}(x,0) = \phi(x). \tag{9}$$

Then we construct a three-dimensional set V by mapping $\mathscr{T}$:

$$V = \{[x,t,\mathscr{T}(x,t) | x \in X, t \in T]\}, \tag{10}$$

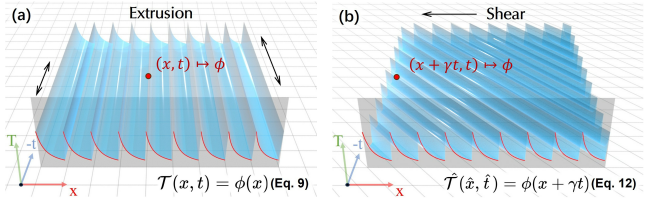


Fig. 6: Pointwise mapping illustration of shearing: a extruded surface and point undergoing a shearing transformation.

whose element v is in the model space $v = \begin{bmatrix} x & t & \mathscr{T}(x,t) \end{bmatrix}^{\mathsf{T}}$. We obtain a new set of elements w by performing temporal shearing transformation on V as shown in Fig. 6a (Γ represents the operator and γ refers to the strength of shearing):

$$w = \Gamma \cdot v = \begin{bmatrix} 1 & \gamma & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ t \\ \mathscr{T}(x,t) \end{bmatrix} = \begin{bmatrix} x+\gamma t \\ t \\ \mathscr{T}(x,t) \end{bmatrix}. \tag{11}$$

w represents the new mapping after shearing, as depicted in Fig. 6b. Through the variational method, the relationship between transparency T, x, and t can be defined as:

$$\hat{\mathscr{T}}(\hat{x},\hat{t}) = \mathscr{T}(x+\gamma t,t) = \phi(x+\gamma t) = \phi\left(\frac{\beta}{k} s(\mathbf{x}_i) + \gamma t\right) \tag{12}$$

7 Exact Apparent Motion

By utilizing the vertex index i as the primary variable, our method implicitly captures the flow map under variable step-size integration via $s(\mathbf{x}_i)$. Consequently, by leveraging the hardware interpolation of $s(\mathbf{p})$ from the vertex to the fragment shader, motion is naturally and precisely embedded into pixel-wise opacity variations. The feasibility of this approach for GPU implementation is illustrated in Fig. 7. At the pixel level, the advection computation can be interpreted as sampling from a pre-defined surface. Each fragment is assigned a identifier $s(\mathbf{p})$, allowing the advection effect to be computed independently at each pixel. This independence eliminates inter-pixel dependencies, which is essential for efficient parallel execution on GPUs. As shown in the t–T planes of Fig. 7, the orange cyclic attenuation curve $T_{s(\mathbf{p}),t}$ encodes the time-varying transparency for each fragment. By applying fixed forward and backward offsets on the t-axis, each parallel computing unit evaluates its own transparency function. The phase differences among these functions naturally produce the spatially modulated cyclic decay described in Sec. 5, enabling both highly parallel computation and perceptually coherent rendering of animated streamlines.

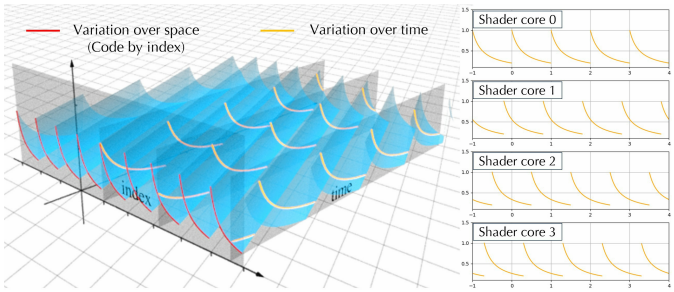


Fig. 7: Principle for GPU implementation: In the fragement shading stage, each computing unit derives its own transparency function, with differences reflected only in their respective phases.

As demonstrated in Fig. 8, mapped color exhibits significant dependence both in time and space. In each computational step, the procedure includes obtaining transparency data from the previous position at the previous time. Then we validate that our animated streamlines accurately reflect variable motion velocities when using an incremental vertex index i. By examining a specific point at a given time, we observe that its value is propagated from the preceding point at the previous time step, aligning precisely with the behavior described by the convection equation:

$$T_{(p_i,s)} = T_{(p_{i-1},s-\Delta s_i)}, \frac{\partial \hat{T}_{(\hat{x},\hat{t})}}{\partial \hat{x}} = -\gamma \frac{\partial \hat{T}_{(\hat{x},\hat{t})}}{\partial \hat{t}}, \tag{13}$$

where x denotes the linear mapping of the vertex index. Let δ_i represent the discrete elements of the index i. For each pair of consecutive points, let Δs_i, denote the time interval and Δ_{r_i} the arc length, which is distinct from the index-based length $\Delta_{\hat{x}}$, we then have:

$$\Delta T_i^1 = \frac{\partial \hat{T}_i}{\partial \hat{x}_i} \delta_i = \frac{\partial \hat{T}_i}{\partial r_i} \frac{\partial r_i}{\partial i} \delta_i \frac{\partial i}{\partial \hat{x}_i} = \frac{\partial \hat{T}_i}{\partial r_i} \Delta_{r_i} \frac{k}{s'(\mathbf{x}_i)\beta}, \tag{14}$$

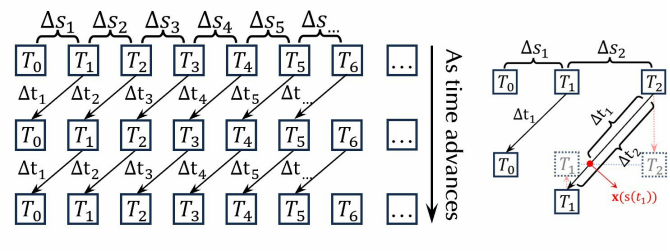


Fig. 8: Transparency computation for streamline points, propagated from the adjacent preceding point and the previous time step to capture spatiotemporal dynamics.

$$\Delta T_i^2 = -\gamma \frac{\partial \hat{T}_i}{\partial \hat{t}_i}\delta_i = -\gamma \frac{\partial \hat{T}_i}{\partial r_i}\frac{\partial r_i}{\partial i}\delta_i \frac{\partial i}{\partial \hat{t}_i} = -\gamma \frac{\partial \hat{T}_i}{\partial r_i}\Delta_{r_i}\frac{\partial i}{\partial \hat{t}_i}. \quad (15)$$

ΔT_i^1 and ΔT_i^2 represent the temporal and spatial step sizes, corresponding to the partial derivatives with respect to time and space, respectively, at the ith step. Note that these values are equal. By equating the two equations, we eliminate Δ_{r_i} and derive the discrete time interval Δs_i between two points as follows:

$$\Delta \hat{t}_i = \frac{\partial \hat{t}_i}{\partial i}\delta_i = -\frac{s'(\mathbf{x}_i)\beta}{\gamma k}\delta_i = -\frac{\lambda_i \beta}{\gamma k}. \quad (16)$$

Given that β, γ, and k are all constants, we can conclude that $\Delta \hat{t}_i$ is positively correlated with λ_i. This essentially represents a velocity scaling factor in Eq. (1), which corresponds to the variable step size increment Δ_s in Runge-Kutta methods as specified in Eq. (4). In polyline-based representations, although step size parameters can only be stored discretely at vertex locations, the standard graphics rendering pipeline supports continuous interpolation of vertex attributes during the fragment stage. This enables smooth variation of the integration step size across fragments. As illustrated in the accompanying figure in Fig. 8, during animation with a fixed time interval Δt_1, the T_2 vertex, though theoretically requiring Δt_2 to reach the position of the T_1 vertex, advances to an intermediate position after a duration of Δt_1. This interpolation-based mechanism effectively resolves inconsistencies in step size progression across vertices that are inherent in variable-step integration algorithms.

Thus, we have completed the mathematical formulation of our animation framework and verified its physical consistency. The derivation shows that motion mapping must be expressed as an integral of the integration step size, rather than simplified variables such as vertex index or local velocity. In modern adaptive integration schemes, the step size Δs can vary by more than an order of magnitude to accommodate grid resolution and flow complexity. With additional arc-length constraints, Δs becomes a highly dynamic quantity, changing abruptly along streamlines. Conventional mappings fail to capture these variations, leading to noticeable perceptual distortions.

As illustrated in the top-down animation of wind flow through an urban building complex (Fig. 9 and the supplemental materials), our time-step-aware mapping produces smooth and physically consistent motion. Streamlets move around buildings with coherent and plausible velocities. In contrast, index- or velocity-based mappings exhibit clear artifacts: streamlets appear to undergo a *phantom deceleration* near building facades, causing unnatural clustering, while high-speed bypass flows are incorrectly perceived as slow.

These artifacts are especially problematic in XR environments, where users are highly sensitive to motion inconsistencies. For fluid dynamics experts, such errors are not merely visual imperfections but sources of misleading interpretation. For instance, apparent upstream deceleration may be misinterpreted as stagnation induced by pressure-coupling errors, while reduced lateral flow can resemble the effects of excessive numerical viscosity near boundaries. By ensuring exact apparent motion, our method removes these misleading effects and enables more reliable analysis of flow behavior.

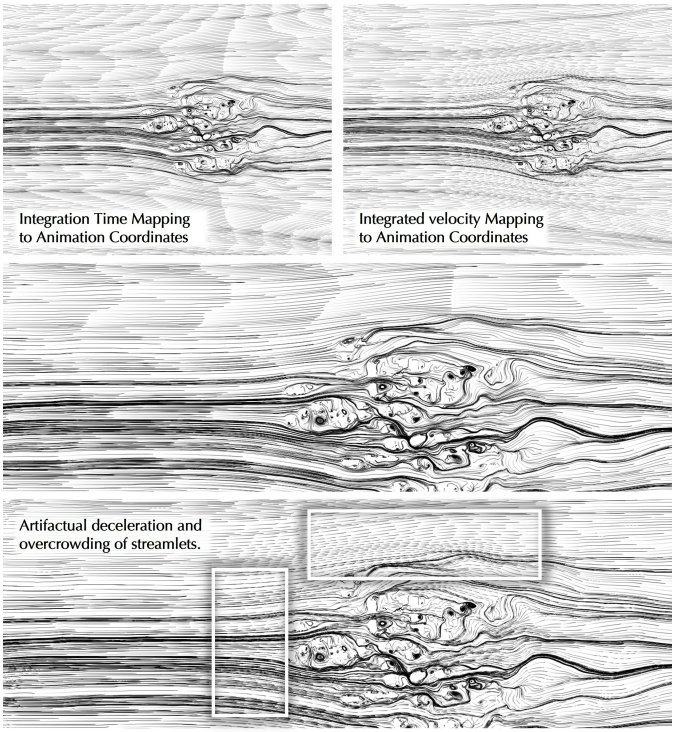


Fig. 9: Visual uncertainty introduced by different mapping variables. Specifically, the speed of apparent motion is positively correlated with streamlet length. Notably, in the boxed region, streamlets shorten and decelerate, leading to local accumulation; this introduces a perceptual distortion that results in an erroneous judgment of velocity. (Please refer to the supplemental video for a more compelling demonstration.)

8 Eulerian Flow Maps

As previously discussed, motion mapping is inherently applied at the vertices of the streamlines; nevertheless, the resulting animation remains continuous within the line segments. This continuity is achieved by deferring the final computation of $\hat{\mathcal{T}}(\hat{x},\hat{t})$ to the fragment shader. As illustrated in the pipeline diagram, we input the integration time to the GPU via a custom Vertex Buffer Object (VBO). During the vertex shading stage, no computation alters this value; it is passed directly to the fragment shader, where the hardware automatically performs linear interpolation across the pixels (leveraging standard rasterization modes like perspective-correct interpolation, nointerpolation, or centroid sampling). Consequently, each pixel within a polyline segment undergoes independent linear interpolation, yielding a constant velocity along that segment that changes abruptly only upon reaching the next vertex. Strictly speaking, the displacement function with respect to time is only C^0 continuous. However, because a single streamlet typically spans hundreds of line segments, these internal velocity discontinuities are visually negligible, manifesting merely as artifacts of geometric discretization. Importantly, we still guarantee that the advection reaching these Eulerian vertices remains physically plausible. This methodology extends naturally to 3D surfaces, provided they encode the integration time attribute. For example, streamlines can be extruded into streamtubes, where the integration time is automatically interpolated across the tube's surface. By adjusting the parameters detailed in 5, we can manipulate animation effects (such as length, phase, and advection speed) in an identical manner, shown in Figs. 10a and 10b.

The framework of "Eulerian Flow Maps" implies that the underlying graphical primitives do not undergo true spatial displacement. To better illustrate this concept and extend our theoretical model, we introduce an alternative rendering pipeline. As depicted in the second flowchart, we can compute $\hat{\mathcal{T}}(\hat{x},\hat{t})$ directly within the vertex shader. By leveraging GPU instancing, we can render massive numbers of glyphs and map

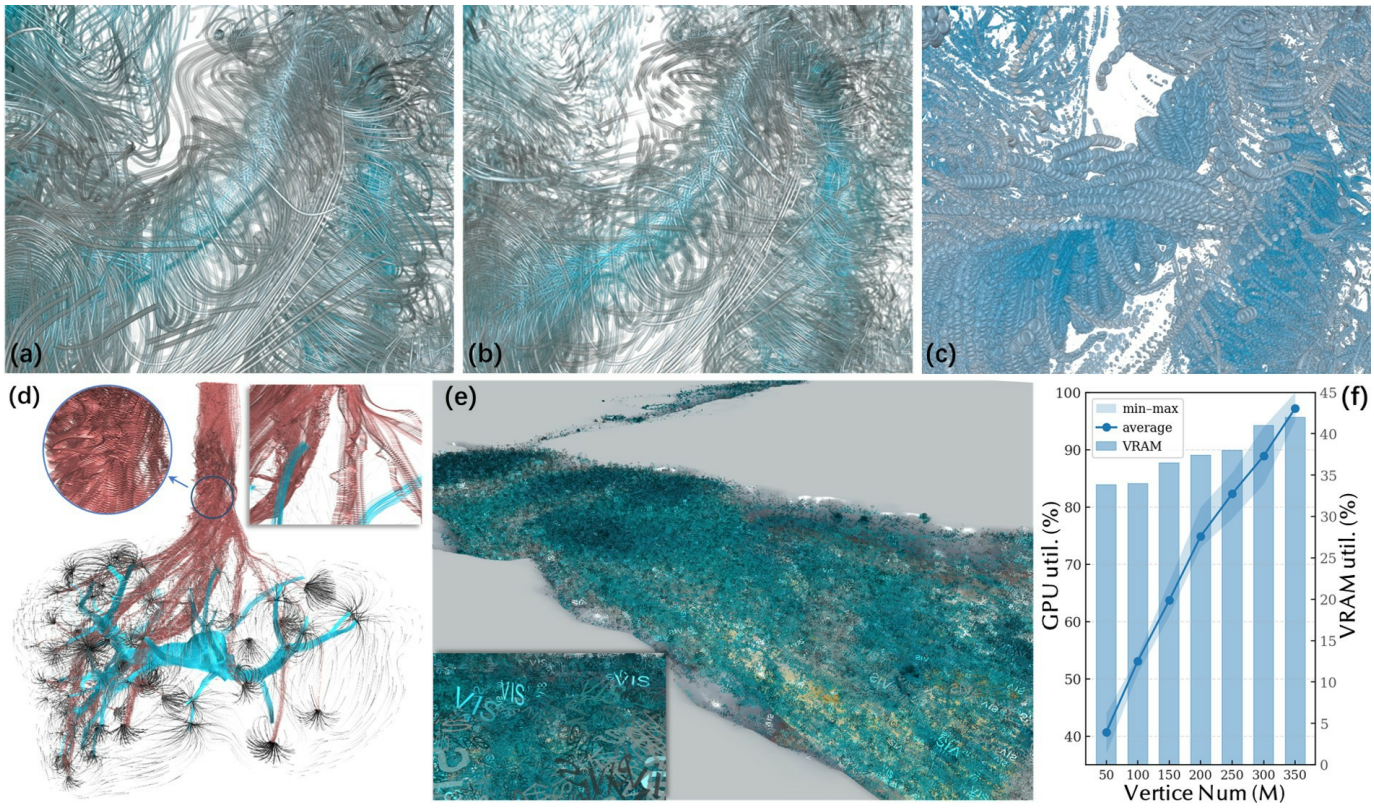


Fig. 10: Various forms of apparent motion based on Eulerian flow maps, fully exploiting GPU parallelism.

$\hat{\mathcal{T}}$ to arbitrary visual channels, such as scale, opacity, and rotation. For instance, as shown in Fig. 10c, we can instance a cell geometry at each streamline vertex and dynamically adjust its size based on the temporal evolution of $\hat{\mathcal{T}}$ to simulate flow. In this scenario, the continuous streamlet is replaced by a sequence of cells with decreasing scales. Modulated by the periodic modulo operation, this creates the visual illusion of conical cell clusters propelling forward. Furthermore, we can instance arrows at the vertices and bypass the modulo function entirely, directly mapping $s(\mathbf{x}_i)$ to a rotation angle. As shown in the red arterial structure of the liver model in Fig. 10d, a helical motion propagates downward; macroscopically, this massive transport looks like cascading blood flow. Yet, under local magnification, it becomes clear that the individual arrows possess zero translational displacement and they merely execute a predefined mechanical rotation in place. We can also introduce more chaotic visualizations by mapping $s(\mathbf{x}_i)$ through a randomized function that simultaneously dictates rotation and scaling. As demonstrated in Fig. 10e, we instanced the characters "VIS" across 0.8 million vertices, producing a dense, animated cascade of text flowing seamlessly through the bathymetry of the Red Sea [16, 45].

Our GPU instancing approach is highly efficient for rendering massive particle datasets. To assess this, we instanced geometric primitives of varying complexities (ranging from 50 to 350 vertices per primitive) across 1M streamline vertices. As shown in Fig. 10f, increasing the geometric fidelity causes GPU utilization to grow linearly, while VRAM consumption remains largely unaffected. This confirms that the rendering bottleneck is primarily bound by GPU compute cores rather than memory bandwidth. In Fig. 10c, we demonstrate this using a 114-vertex cell geometry with optimized normals to achieve a smooth appearance. To ensure visual continuity during animation, the spatial gap between instanced vertices must be kept small. We therefore recommend a fixed-arc-length RK4 integration scheme based on normalized velocity [29]. This guarantees equidistant streamline vertices and provides the necessary point density for instancing. Throughout our benchmark tests, performance hovered reliably around 24 FPS, note that occasional system-level stuttering can temporarily drop the frame rate, leading to a corresponding decrease in GPU utilization. Overall, these results validate that our Eulerian flow map method is exceptionally well-suited for dense, streamline-based particle visualization.

9 Interactive Latency

Our animated streamline framework balances numerical accuracy from integration methods with high performance for interactive use. As outlined in Sec. 4, streamline generation is accelerated through a multi-stage pipeline. To assess scalability, we evaluated three parallelization strategies implemented in VTK. The test dataset was an urban street wind simulation with 11,567,302 cells (Detail in Appx. A). Experiments were run on an Intel i9-13900KF CPU with 32 cores. As shown in Fig. 11, peak performance occurs when the number of threads matches the physical core count. The TBB (Threading Building Blocks) method, which leverages hyper-threading to assign multiple threads per core, suffers a sharp performance drop at 128 threads due to oversubscription. In contrast, the other methods cap thread usage at the hardware limit. Among all tested techniques, TBB delivered the best results, outperforming OpenMP by more than 70% under optimal conditions.

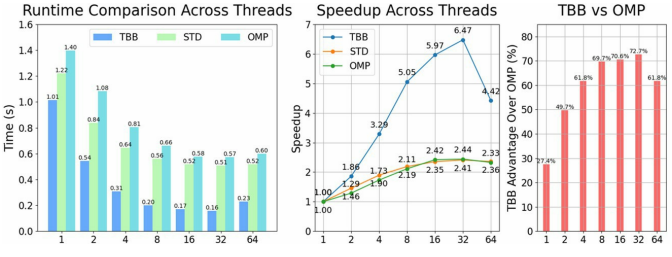


Fig. 11: Comparison and analysis of strong scalability among different parallel implementation techniques.

We further examined the scalability of our streamline generation algorithm by averaging five runs for each test case (Fig. 12). The smallest test generated 512 streamlines (about 10 million points), while the largest produced 8192 streamlines (about 160 million points). The left panel shows execution time decreasing consistently as core counts increase from 2 to 64, confirming strong scalability. For instance, generating 8192 streamlines dropped from roughly 40 seconds to under 10 seconds. The red dashed line with star markers indicates theoretical weak scalability, where runtime would increase only modestly with more cores if workload per core remains constant. The right panel of Fig. 12 reports parallel efficiency, which is highest at low core counts and declines as more cores are added due to the growing impact of non-parallelizable operations. Nevertheless, overall speedup remains significant, underscoring the algorithm's scalability and suitability for real-time visualization. Note that even the smallest case, 10 million points, which is already quite dense (see in Appx. B), generated in just 0.57 seconds, far exceeding the typical need of only a few thousand points per interaction.

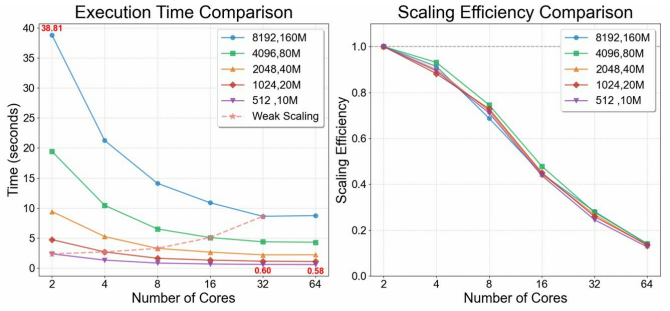


Fig. 12: Scalability and Efficiency of Streamline Generation.

10 Animation Efficiency

Current animated streamline techniques remain basic. For instance, ParaView introduced an image-based method in 2017 that accumulates particle tracks in a 2D buffer to record trajectory histories. While this enables simple lifecycle tracking, it has critical drawbacks: **1). Scene Integration:** Operating in 2D image space prevents seamless integration with 3D environments. Streamlines appear as overlays above 3D objects, disrupting visual coherence. **2). Camera Flexibility:** The image-blending approach assumes a fixed camera view. Changing viewpoints requires recomputation of trajectories, making it unsuitable for dynamic cameras, such as those in head-mounted displays. **3) Computational Inefficiency:** As a Lagrangian method, it requires real-time integration and advection computations at every frame. This continuous re-evaluation fundamentally limits frame rates compared to precomputation-based strategies (a detailed algorithmic and performance comparison is provided in Appx. E).

We also evaluated the scalability of our method using the urban buildings dataset. Benchmarks were run on a desktop with an NVIDIA RTX 4090 GPU at 1K resolution. As shown in Fig. 13, the x-axis represents the number of streamline vertices (in millions), and the y-axis shows frame rate. Our method sustains real-time performance (above 30 FPS) with up to 600 million vertices, confirming that it minimizes rendering overhead while preserving responsiveness for interactive exploration. For higher resolutions, we repeated the benchmarks at 4K (3840×2160) and 8K (7680×4320) resolutions. Frame rates remained stable except at 8K with 600 million vertices, where density-induced overdraw caused a noticeable slowdown. Additional experiments with concentrated and uniform streamline distributions (see Supplementary Materials) revealed similar trends, with higher screen-space density leading to reduced frame rates. We also tested portability on two laptop systems. The red and purple lines in Fig. 13 show performance for an RTX 4090 Laptop and an RTX 3060 Laptop, respectively. Both maintained real-time rendering above 30 FPS even with 100 million vertices. Overall, these results demonstrate that our method delivers scalable, real-time animated streamline rendering across both high-performance desktops and portable platforms, making it well-suited for interactive visualization of large, complex flow datasets.

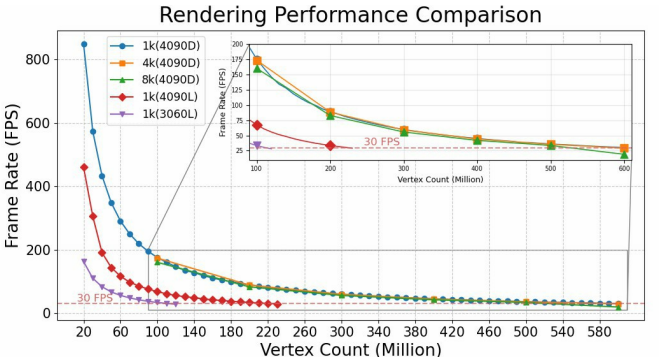


Fig. 13: Efficiency of dense animated streamlines rendering.

11 Graphics Optimization

Our method further optimizes rendering by supporting efficient batch processing of streamlines using structured primitives compatible with modern graphics pipelines. Specifically, streamline segments can be represented as either *Line List* (independent segments) or *Line Strip* (continuous polylines). We adopt the Line List representation, where each pair of vertices defines an independent segment, allowing multiple disjoint streamlines to be generated in parallel. Segment offsets are computed via prefix-sum operations, after which a compute shader dynamically assembles the line primitives in GPU buffers. This enables thousands of independent streamlines to be aggregated into a single batched draw call, avoiding the connectivity constraints of Line Strips. Beyond simplifying memory management and data preparation, Line Lists maximize rendering efficiency by reducing the frequency of CPU–GPU communications.

As illustrated in Fig. 14, we evaluated the impact of batch processing on rendering performance. Tests were conducted on the same desktop equipped with an RTX 4090 GPU at 1K resolution. We progressively batched 2^{26} line segments, from 32,768 LineSets down to a single merged LineSet. Results show that as batch size increases, frame rates improve from approximately 71 FPS to over 261 FPS, achieving more than a 3x performance improvement. These results validate the effectiveness of batching techniques, particularly when processing large-scale streamline data. However, as batch sizes continue to increase, performance gains tend to plateau. Therefore, while batching continues to optimize performance, its marginal benefits diminish progressively. We also tested with a larger number of line segments (410 million), as shown by the blue line. After merging to a critical point, performance drops sharply with no graphics rendered, likely reaching the GPU's limit for processing vertices in a single operation. Through more tests, this limit appears to be between 150M to 160M vertices. Therefore, the relationship between batch size and applications must be considered to avoid negative impacts from excessive batching.

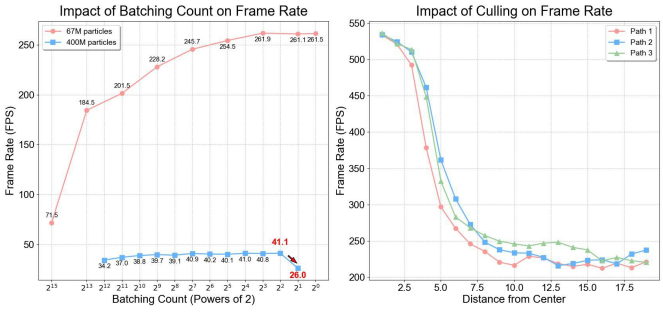


Fig. 14: Impact of batching and culling on rendering performance.

Another key advantage of our approach is its compatibility with culling techniques, yielding substantial performance gains. Because streamline generation is fully parallelized and does not depend on prior frame data, our method supports real-time viewpoint switching while restricting computation exclusively to visible flow regions. Unlike Lagrangian approaches—which track particles along entire streamlines and waste resources on irrelevant areasour GPU-powered solution computes visible segments on the fly. This reduces both computational and rendering latency. The combination of pixel-level parallelism with frustum and occlusion culling minimizes redundant processing and sustains stable frame rates. This design is particularly critical for VR, where massive streamline datasets must be rendered responsively while adapting to localized GPU workloads.

We conducted multi-viewpoint rendering tests to evaluate the effectiveness of this culling strategy. Starting from the center of a flow field populated with animated streamlines (approximately 40M vertices), we incrementally moved the camera outward along fixed paths until the entire dataset came into view. To ensure the generalizability of our results, we tested three distinct paths. As shown in Fig. 14, as the viewpoint recedes and more streamlines enter the field of view, the frame rate gradually decreases. This confirms that our culling approach effectively maintains high performance and maximizes efficiency during the interior exploration of complex flow fields.

12 Interaction Exploration and Animation Control

In our XR platform, users can navigate freely once the scene loads. Upon reaching a region of interest, they can place anchors to seed streamlines. We provide two seeding modes: **Point Source** distributes seed points within a sphere, with a radius adjustable via two-handed scaling. **Line Source** distributes seed points along a segment between two user-specified endpoints. Both modes allow the adjustable seed density, users then may navigate freely, change locomotion speed, and observe the generated streamlines. This combination of flexible navigation and precise seeding supports both rapid exploration and detailed analysis.(More detail is shown in Fig. 18)

The procedure illustrates how our framework supports progressively exploration in flow visualization. Firstly, seeding on the leeward side of a building reveals a spiral structure (Fig. 15a). We then navigate to the structure, placing new seed points to highlight the vortex in greater detail (Fig. 15b). As we explore the base of the building (Fig. 15c), we observe subtle circular flows motion that suggest the potential presence of a vortex. Through progressive seeding, we successfully delineate this vortex using animated streamlines (Fig. 15d). Together, these steps form a systematic approach to analyzing local flow structures. Note that simply adding more streamlines does not necessarily improve clarity, our framework supports a step-by-step rollback of the seeding process to continually refine the analysis. Targeted seeding in Fig. 15c demonstrates three insights: 1) Vortex structures can be detected not only by their geometric morphology but also through the dynamic animation of streamlines. 2) Color mapping of terrain and buildings based on physical properties helps identify velocity gradients that may indicate vortex locations. 3) Groups of streamlines may form enclosing

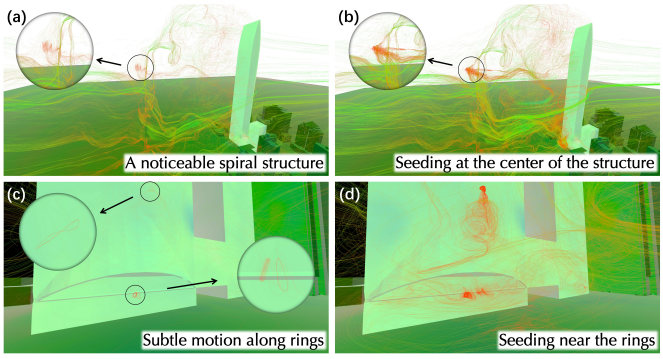


Fig. 15: Progressive Exploration of Flow Structures.

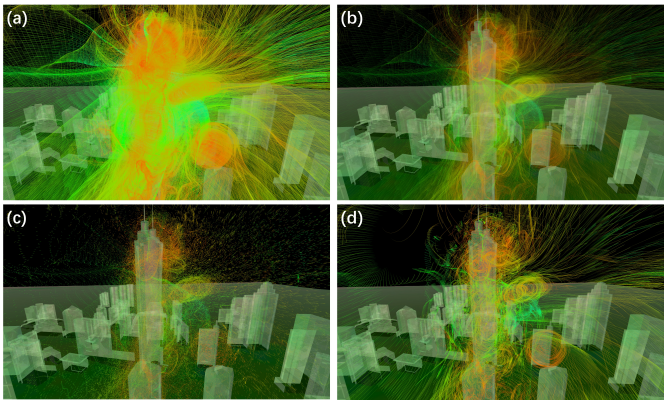


Fig. 16: Diverse Animated Flow Patterns, enlargement is recommended.

envelopes, suggesting the boundary of a vortex. These capabilities allow researchers to interpret vortices from both geometric and dynamic perspectives, adding interpretive depth and enhancing visual analysis.

In Sec. 5, we derive the streamlet-based opacity encoding that enters the discrete transparency model in Eq. (12) along streamlines; parameters α, β/k, τ, and λ control streamlets' shape, segmentation, and decay. In Sec. 6, the shear strength γ sets the global apparent motion speed of the animated streamlets. These controls supports a richer inspection of dynamic visualizations of fluid. Fig. 16a shows a heavily seeded streamline field around the urban model. Modest transparency adjustments greatly reduce visual clutter and expose interior vortex structures in Fig. 16b, drawing attention to swirling regions. This behavior highlights, to some extent, an advantage of dense streamline visualization: trajectories naturally concentrate near vortices, forming salient swirling cores that do not appear with fewer streamlines. Raising transparency while shortening streamlets yields a complementary effect (Fig. 16c): compact streamlets produce a fast, particle-like spray that keeps façade geometry readable while still conveying high-speed transport. Fig. 16d instead uses longer streamlines with wider spacing to limit occlusion among dense trajectories and to uncover interior recirculation and wake motion behind buildings. No single multi-parameter strategy is uniformly optimal across tasks and viewpoints; we therefore expose the full control set so that users can foreground the flow motion pattern most salient to their analysis objectives.

13 Limitations and Conclusion

While our framework advances real-time animated streamline visualization in XR, we acknowledge the absence of a formal user study to evaluate its broader impact on human-centered visualization and interaction. A comprehensive investigation into immersive motion perception, interaction facilitation, and the cognitive benefits of progressive exploration constitutes a substantial and highly nuanced research endeavor in its own right. Given the rigorous qualitative and quantitative methodologies such user-centric evaluations demand, we consider these dimensions to be beyond the scope of the present study, which centers specifically on algorithmic rendering efficiency and the physical fidelity of motion representation. We view these critical efforts as an important and promising direction for future work.

In conclusion, we have presented a novel, GPU-accelerated framework that enables the real-time interaction of hundreds of millions of dynamic streamlets within extended reality. By introducing Eulerian Flow Maps coupled with spacetime shear transformations, our approach successfully decouples spatial continuity from temporal progression, guaranteeing an exact apparent motion that strictly aligns with underlying physical velocities. This methodology effectively eliminates the visual artifacts inherent in variable-step integration and overcomes the computational bottlenecks of traditional advection paradigms. Ultimately, this work delivers a scalable, physically faithful, and interactive visualization tool that empowers researchers to intuitively decipher complex fluid dynamics.

Supplemental Materials

All supplementary materials are available at https://reshyx.github.io/Dense-Animated-Streamlines/, they include:

- A teaser video for some motivations behind our work.
- Panoramic videos showcasing the immersive interaction.
- WebXR implementations for animated streamline visualization, with adjustable parameters in the browser. Highly recommended and convenient to try it out.
- A Quest application for demonstration of our interactions, (supporting fewer streamlines and glyphs due to device limitations).
- A desktop graphics testing application to validate performance.
- ParaView plugins for animated streamline and glyph visualization, with more parameters and features to research.
- Additional application examples demonstrating diverse use cases of our visualization system.

References

[1] U. Ayachit. *The paraview guide: A parallel visualization application*. Kitware, Inc., 2015. 2

[2] S. Bachthaler and D. Weiskopf. Animation of orthogonal texture patterns for vector field visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):741–755, July 2008. doi: 10.1109/TVCG.2008.36 2

[3] N. Bhatia and O. K. Matar. Learning and Teaching Fluid Dynamics using Augmented and Mixed Reality. In *2022 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 865–869, Oct. 2022. 1

[4] R. Botchen, D. Weiskopf, and T. Ertl. Texture-based visualization of uncertainty in flow fields. In *VIS 05. IEEE Visualization, 2005.*, pp. 647–654, 2005. 2

[5] J. R. Cash and A. H. Karp. A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides. *ACM Transactions on Mathematical Software (TOMS)*, 16(3):201–222, 1990. 3

[6] Y. Cen, H. Deng, Y. Ma, and X. Liang. A Real-Time and Interactive Fluid Modeling System for Mixed Reality. *IEEE Transactions on Visualization and Computer Graphics*, 30(11):7310–7320, Nov. 2024. 1

[7] W. Chen, T. Sang, Y. Ma, Q. Chen, Y. Xiao, Z. Pan et al. Real-time simulation of violent boiling in concentrated sulfuric acid dilution. *The Visual Computer*, 37(9-11):2631–2642, Sept. 2021. 1

[8] M. Cox and D. Ellsworth. Managing big data for scientific visualization. *Proceedings of ACM Siggraph, Ames: NASA*, pp. 21–38, 1997. 1

[9] Y. Ding and L. Harrison. Data in the wind: Evaluating multiple-encoding design for particle motion visualizations. In *2023 IEEE Visualization and Visual Analytics (VIS)*, pp. 151–155, Oct 2023. doi: 10.1109/VIS54172.2023.00039 2

[10] C. Donalek, S. G. Djorgovski, A. Cioc, A. Wang, J. Zhang, E. Lawler et al. Immersive and collaborative data visualization using Virtual Reality platforms. In *2014 IEEE International Conference on Big Data (Big Data)*, pp. 609–614. IEEE, 2014. doi: 10.1109/BigData.2014.7004282 2

[11] S. Eichelbaum, M. Hlawitschka, and G. Scheuermann. LineAO—Improved Three-Dimensional Line Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):433–445, 2013. 2

[12] S. Eun, T. Noh, and M. Lee. Effects of Peripheral Optic Flow Location and Speed on Unintended Positional Drift During Walk-in-Place in VR. In *2025 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 1501–1511. IEEE, 2025. 2

[13] C. Garth and K. I. Joy. Fast, Memory-Efficient Cell Location in Unstructured Grids for Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1541–1550, 2010. 3

[14] C. F. Gerald. *Applied numerical analysis*. Pearson Education India, 2004. 3

[15] D. Groß and S. Gumhold. Advanced Rendering of Line Data with Ambient Occlusion and Transparency. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):614–624, 2021. 2

[16] I. Hoteit, T. Hoar, G. Gopalakrishnan, N. Collins, J. Anderson, B. Cornuelle et al. A MITgcm/DART ensemble analysis and prediction system with application to the Gulf of Mexico. *Dynamics of Atmospheres and Oceans*, 63:1–23, 2013. 7

[17] M. Ikits. *Interactive exploration of volumetric data sets with a combined visual and haptic interface*. University of Utah, 2007. 1

[18] B. Jobard and W. Lefer. The motion map: Efficient computation of steady flow animations. In *Proceedings. Visualization '97 (Cat. No. 97CB36155)*, pp. 323–328, Oct 1997. doi: 10.1109/VISUAL.1997.663899 2, 4

[19] B. Jobard and W. Lefer. Unsteady flow visualization by animating evenly-spaced streamlines. In *Computer Graphics Forum*, vol. 19, pp. 31–39. Wiley Online Library, 2000. doi: 10.1111/1467-8659.00395 2, 12

[20] M. Kern, C. Neuhauser, T. Maack, M. Han, W. Usher, and R. Westermann. A Comparison of Rendering Techniques for 3D Line Sets With Transparency. *IEEE Transactions on Visualization and Computer Graphics*, 27(8):3361–3376, 2021. 2

[21] D. Ketcheson and U. Bin Waheed. A comparison of high-order explicit Runge–Kutta, extrapolation, and deferred correction methods in serial and parallel. *Communications in Applied Mathematics and Computational Science*, 9(2):175–200, 2014. 4

[22] A. J. F. Kok and R. Van Liere. A multimodal virtual reality interface for 3D interaction with VTK. *Knowledge and Information Systems*, 13(2):197–219, Oct. 2007. 2

[23] E. H. Korkut and E. Surer. Visualization in Virtual Reality: A systematic review. *Virtual Reality*, 27(2):1447–1480, 2023. doi: 10.1007/s10055-023-00753-8 2

[24] E.-C. Lee, Y.-H. Cho, and I.-K. Lee. Simulating Water Resistance in a Virtual Underwater Experience Using a Visual Motion Delay Effect. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 259–266. IEEE, Osaka, Japan, Mar. 2019. 1

[25] W. Lefer, B. Jobard, and C. Leduc. High-quality animation of 2d steady vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):2–14, 2004. 2, 4

[26] J. Leigh, P. J. Rajlich, R. J. Stein, A. E. Johnson, and T. A. DeFanti. LIMBO/VTK: A tool for rapid tele-immersive visualization. In *IEEE Visualizaton*, pp. 18–23, 1998. 2

[27] S. Leveque, L. Bergamaschi, Á. Martínez, and J. W. Pearson. Parallel-in-time solver for the all-at-once runge-kutta discretization. *SIAM Journal on Matrix Analysis and Applications*, 45(4):1902–1928, 2024. 4

[28] X. Liu, Y. Liu, and Y. Wang. Real time 3D Magnetic Field Visualization Based on Augmented Reality. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 1052–1053, March 2019. doi: 10.1109/VR.2019.8797782 2

[29] Z. Liu, R. Moorhead, and J. Groner. An Advanced Evenly-Spaced Streamline Placement Algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):965–972, 2006. 7, 12

[30] M.-J. Lobo, A. C. Telea, and C. Hurter. Feature driven combination of animated vector field visualizations. In *Computer Graphics Forum*, vol. 39, pp. 429–441. Wiley Online Library, 2020. 2

[31] S. K. Lodha, A. Pang, R. E. Sheehan, and C. M. Wittenbrink. Uflow: Visualizing uncertainty in fluid flow. In *Proceedings of Seventh Annual IEEE Visualization'96*, pp. 249–254. IEEE, 1996. 1, 2, 3

[32] Q. Ma, J. Wu, R. Fan, G. Sun, and X. Shi. ViP-Fluid: Visual Perception Driven Method for VR Fluid Rendering. In *2024 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 359–367, Oct. 2024. 1

[33] H. Matovu, D. A. K. Ungu, M. Won, C.-C. Tsai, D. F. Treagust, M. Mocerino et al. Immersive Virtual Reality for science learning: Design, implementation, and evaluation. *Studies in Science Education*, pp. 1–40, 2023. 2

[34] O. Mattausch, T. Theußl, H. Hauser, and E. Gröller. Strategies for interactive exploration of 3D flow using evenly-spaced illuminated streamlines. In *SCCG03: Spring Conference in Computer Graphics 2003*, pp. 213–222. ACM, 2003. 2

[35] N. Max and B. Becker. Flow visualization using moving textures. In *NASA Conference Publication*, pp. 77–88. NASA, 1996. 4

[36] R. Mintz, S. Litvak, and Y. Yair. 3D-Virtual Reality in science education: An implication for astronomy teaching. *Journal of Computers in Mathematics and Science Teaching*, 20(3):293–305, 2001. 2

[37] O. Mishchenko and R. Crawfis. On Perception of Semi-Transparent Streamlines for Three-Dimensional Flow Visualization. *Computer Graphics Forum*, 33(1):210–221, 2014. 2

[38] R. A. Montano-Murillo, C. Nguyen, R. H. Kazi, S. Subramanian, S. DiVerdi, and D. Martinez-Plasencia. Slicing-Volume: Hybrid 3D/2D Multi-target Selection Technique for Dense Virtual Environments. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 53–62, 2020. 1

[39] L. Murray. GPU Acceleration of Runge-Kutta Integrators. *IEEE Transactions on Parallel and Distributed Systems*, 23(1):94–101, 2012. 4

[40] G. L. Naber. *The geometry of Minkowski spacetime*. Springer, 2012. 5

[41] B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen. Load-Balanced Parallel Streamline Generation on Large Scale Vector Fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1785–1794, 2011. 4

[42] M. OĆonnor, H. M. Deeks, E. Dawn, O. Metatla, A. Roudaut, M. Sutton et al. Sampling molecular conformations and dynamics in a multiuser Virtual Reality framework. *Science Advances*, 4(6):eaat2731, 2018. doi: 10.1126/sciadv.aat2731 2

[43] P. O'Leary, S. Jhaveri, A. Chaudhary, W. Sherman, K. Martin, D. Lonie et al. Enhancements to VTK enabling scientific visualization in immersive environments. In *2017 IEEE Virtual Reality (VR)*, pp. 186–194. IEEE, Los Angeles, CA, USA, 2017. 1, 2

[44] J. Quarles. Shark punch: A virtual reality game for aquatic rehabilitation. In *2015 IEEE Virtual Reality (VR)*, pp. 375–375. IEEE, Arles, Camargue, Provence, France, Mar. 2015. 1

[45] S. Reddy, H. Toye, P. Zhan, S. Langodan, G. Krokos, O. Knio et al. Impact of atmospheric and model physics perturbations on a high-resolution ensemble data assimilation system of the red sea. *Journal of Geophysical Research: Oceans*, 125, 08 2020. doi: 10.1029/2019JC015611 7

[46] B. E. Riecke and J. D. Jordan. Comparing the effectiveness of different displays in enhancing illusions of self-movement (vection). *Frontiers in Psychology*, 6:713, 2015. 2

[47] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko. Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1325–1332, Nov 2008. doi: 10.1109/TVCG.2008.125 2, 4

[48] S. Sane, R. Bujack, C. Garth, and H. Childs. A Survey of Seed Placement and Streamline Selection Techniques. *Computer Graphics Forum*, 39(3):785–809, 2020. 2

[49] M. Schirski, C. Bischof, and T. Kuhlen. Interactive exploration of large data in hybrid visualization environments. EGVE'07, pp. 69–76. Eurographics Association, Goslar, DEU, 2007. 1

[50] M. Schirski, T. Kuhlen, M. Hopp, P. Adomeit, S. Pischinger, and C. Bischof. Efficient visualization of large amounts of particle trajectories in virtual environments using virtual tubelets. In *The 2004 ACM SIGGRAPH International Conference*, p. 141. ACM Press, 2004. 1

[51] W. J. Schroeder, L. S. Avila, and W. Hoffman. Visualizing with VTK: A tutorial. *IEEE Computer Graphics and Applications*, 20(5):20–27, 2000. doi: 10.1109/38.865875 2

[52] T. Seno, K. Murata, Y. Fujii, H. Kanaya, M. Ogawa, K. Tokunaga et al. Vection is enhanced by increased exposure to optic flow. *i-Perception*, 9(3):1–16, May 2018. doi: 10.1177/2041669518774069 2

[53] H.-W. Shen and D. Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):98–108, April 1998. doi: 10.1109/2945.694952 2

[54] R. G. Shoup. Color table animation. In *Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 8–13, 1979. 4

[55] J. N. Silva, M. Southworth, C. Raptis, and J. Silva. Emerging applications of Virtual Reality in cardiovascular medicine. *JACC: Basic to Translational Science*, 3(3):420–430, 2018. doi: 10.1016/j.jacbts.2017.11.009 2

[56] L. Silvestri. CFD modeling in industry 4.0: New perspectives for smart factories. *Procedia Computer Science*, 180:381–387, 2021. doi: 10.1016/j.procs.2021.01.359 2

[57] R. Simpson, J. LaViola, D. Laidlaw, A. Forsberg, and A. Van Dam. Immersive VR for scientific visualization: A progress report. *IEEE Computer Graphics and Applications*, 20(6):26–52, 2000. doi: 10.1109/38.888006 2

[58] S. Solmaz and T. Van Gerven. Automated integration of extract-based CFD results with AR/VR in engineering education for practitioners. *Multimedia Tools and Applications*, 81(11):14869–14891, 2022. doi: 10.1007/s11042-021-10621-9 2

[59] D. Stalling. *Fast texture based algorithms for vector field visualization*. PhD thesis, Free University of Berlin, Dahlem, Germany, 1999. 2

[60] S. Su, I. Lopez-Coto, W. R. Sherman, K. Sayrafian, and J. Terrill. Immersive ParaView: An Immersive Scientific Workflow for the Advancement of Measurement Science. In *2022 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 139–145. IEEE, Singapore, Singapore, Oct. 2022. 1

[61] A. Telea and J. van Wijk. 3D IBFV: hardware-accelerated 3D flow visualization. In *IEEE Visualization, 2003. VIS 2003.*, pp. 233–240, Oct 2003. doi: 10.1109/VISUAL.2003.1250377 2

[62] M. Uda and T. Nakamoto. Influence of User's Body in Olfactory Virtual Environment Generated by Real-Time CFD. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pp. 951–959, Mar. 2024. 1

[63] S.-K. Ueng, C. Sikorski, and K.-L. Ma. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):370–380, 1997. 4

[64] Utkarsh, C. Elrod, Y. Ma, K. Althaus, and C. Rackauckas. Parallelizing Explicit and Implicit Extrapolation Methods for Ordinary Differential Equations. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–9, 2022. 4

[65] A. Van Gelder and J. Wilhelms. Interactive visualization of flow fields. In *Proceedings of the 1992 Workshop on Volume Visualization*, pp. 47–54, 1992. 2

[66] T. van Reimersdahl, T. Kuhlen, A. Gerndt, J. Henrichs, and C. Bischof. ViSTA: A multimodal, platform-independent VR-Toolkit based on WTK, VTK, and MPI. In *Proceedings of the 4th International Immersive Projection Technology Workshop*, vol. 14, p. 22, 2000. 2

[67] J. J. Van Wijk. Image based flow visualization. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 745–754, 2002. 2

[68] M. Venkatesan, H. Mohan, J. R. Ryan, C. M. Schürch, G. P. Nolan, D. H. Frakes et al. Virtual and augmented reality for biomedical applications. *Cell Reports Medicine*, 2(7), 2021. doi: 10.1016/j.xcrm.2021.100348 2

[69] Y. Wang, Y. Zhang, X. Yang, H. Wang, D. Liu, and X. Yang. Foveated Fluid Animation in Virtual Reality. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pp. 535–545, Mar. 2024. 1

[70] C. Ware, D. Bolan, R. Miller, D. H. Rogers, and J. P. Ahrens. Animated versus static views of steady flow patterns. In *Proceedings of the ACM Symposium on Applied Perception*, pp. 77–84. ACM, 2016. doi: 10.1145/2931002.2931012 2, 4

[71] R. Wegenkittl and E. Groller. Fast oriented line integral convolution for vector field visualization via the internet. In *Proceedings. Visualization '97 (Cat. No. 97CB36155)*, pp. 309–316, Oct 1997. doi: 10.1109/VISUAL.1997.663897 2

[72] R. Wegenkittl, E. Groller, and W. Purgathofer. Animating flow fields: Rendering of oriented line integral convolution. In *Proceedings. Computer Animation '97 (Cat. No.97TB100120)*, pp. 15–21, June 1997. doi: 10.1109/CA.1997.601035 2, 4

[73] D. Weiskopf, T. Schafhitzel, and T. Ertl. Texture-Based Visualization of Unsteady 3D Flow by Real-Time Advection and Volumetric Illumination. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):569–582, 2007. 2

[74] D. Weiskopf, F. Schramm, G. Erlebacher, and T. Ertl. Particle and texture based spatiotemporal visualization of time-dependent vector fields. In *VIS 05. IEEE Visualization, 2005.*, pp. 639–646, Oct 2005. doi: 10.1109/VISUAL.2005.1532852 2

[75] G. Wheeler, S. Deng, N. Toussaint, K. Pushparajah, J. A. Schnabel, J. M. Simpson et al. Virtual interaction and visualisation of 3D medical imaging data with VTK and Unity. *Healthcare Technology Letters*, 5(5):148–153, Sept. 2018. 2

[76] M. Wierzbicki, M. Drangova, G. Guiraudon, and T. Peters. Validation of dynamic heart models obtained using non-linear registration for Virtual Reality training, planning, and guidance of minimally invasive cardiac surgeries. *Medical Image Analysis*, 8(3):387–401, 2004. doi: 10.1016/j.media.2004.06.014 2

[77] J. Yan, K. Kensek, K. Konis, and D. Noble. CFD visualization in a Virtual Reality environment using building information modeling tools. *Buildings*, 10(12):229, 2020. doi: 10.3390/buildings10120229 2

[78] C.-K. Yeh, Z. Liu, and T.-Y. Lee. Animating streamlines with repeated asymmetric patterns for steady flow visualization. In *Visualization and Data Analysis 2012*, vol. 8294, pp. 286–297. SPIE, 2012. 2

[79] J. Zhang, H. Guo, F. Hong, X. Yuan, and T. Peterka. Dynamic Load Balancing Based on Constrained K-D Tree Decomposition for Parallel Particle Tracing. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):954–963, 2018. 4

[80] K. Zheng, Y. Ci, H. Liu, and J. Zhang. A spatio-temporal process visualization approach for wind features. *Computational Geosciences*, 25:2055–2067, 2021. 2

A Dataset Sources, Scale, and Seeding

Our experiments draw on five volumetric flow fields from different application settings: pollutant transport from a nuclear facility in an urban Bay Area model; meter-scale, pedestrian-height wind through building-lined streets; eddy-dominated transport in a Red Sea circulation dataset; multi-region hepatic flow across tissue and major vessels (vein, portal vein, and artery); and turbulent flow in a model cardiac cavity. They span large environmental simulations to organ-scale hemodynamics and stress our pipeline at disparate resolutions and topologies. Table 1 records, for each case, cell and vertex counts (in millions), the dominant polyhedral type, and which seeding code(s) apply. Streamline seeding is grouped into categories **S1**–**S4** as follows.

- **S1:** Manual exploration in XR: regions-of-interest marking and insight-driven seed placement on discovered vortical structures.
- **S2:** Evenly-spaced streamline placement [19, 29], followed by unrestricted RK45 integration.
- **S3:** Vorticity-adaptive seed placement using the λ_2 criterion.
- **S4:** Seeds distributed on vessel inlet/outlet surfaces.

Table 1: Per-dataset cell and vertex counts (millions), dominant polyhedral type, and seeding code(s). Codes S1–S4 are defined above.

Dataset	Cells (M)	Verts (M)	Polyhedra	Seed
Nuclear plume (Bay Area)	19.97	3.72	Hexahedral	S1
Urban street wind	11.57	7.32	Tetrahedral	S2
Red Sea eddy transport	4.03	4.32	Cartesian	S3
Liver multi-region flow	11.64	2.27	Tetrahedral	S1, S4
Heart cavity turbulence	2.84	0.59	Tetrahedral	S1, S3

B Interactive Latency

This appendix continues the timing discussion in Sec. 9 and Fig. 12. Our smallest timed run used 512 streamlines, about 10^7 polyline vertices in total, and finished in about 0.57 s. In normal use, one interaction only needs a few thousand integrated points for a clear picture on screen. So capping a single step at about ten million vertices is a large safety margin: generation stays well under one second on our setup (Fig. 12), but that cap is far above what one click or one seeding action usually needs. Figure 17 supplies a visual counterpart. Each panel uses *strictly fewer* than ten million streamline vertices, yet the bundles already appear thick and strongly occluding at normal viewing scales. We include this figure so readers can judge density directly. Taken together with Fig. 12, the message is twofold: interactive generation can comfortably target well below 10^7 vertices per step while still looking saturated, and even those sub-10^7 configurations are visually very dense.

C XR Locomotion and Seeding Interface

Immersive flow exploration combines fast locomotion with precise seeding. The main text summarizes point-source and line-source modes; Fig. 18 collects six head-mounted snapshots from a typical session ((a)–(f)). The following paragraphs explain each pair of panels.

Locomotion and flow field context (Figs. 18a and 18b). Figure 18a shows live horizontal and vertical locomotion speeds together with schematic controllers: thumbsticks adjust travel speed along each axis so users can traverse large sites quickly and then slow down for precise placement, reaching intended seeding locations efficiently. Figure 18b shows two controls that shape flow-field exploration: the terrain colormap encodes the spatially varying scalar field, and seed-source mode selects how seeds are issued (e.g., point versus line-driven seeding), which together govern how and where users place seeds.

Endpoint placement for line sources (Figs. 18c and 18d). Line-source seeding requires aiming controller rays at geometry or proxy objects. Figure 18c shows controller-based ray interaction with a proxy sphere (green hit disk); the same affordance supports gesture-based input for repositioning and resizing the proxy sphere. Figure 18d shows a guided sequence for novice users: after placing the first line endpoint, the user spawns a spherical handle for the second endpoint, drags it toward a target, and releases the pinch button to confirm that endpoint.

Integrated streamlines (Figs. 18e and 18f). After seeds are committed, Fig. 18e depicts dense streamlines together with a controller map (buttons, trackpad, trigger, grip) for reference, while color-mapped animated streamlines are already visible in the headset. Figure 18f focuses on the animated streamline visualization with instructional panels hidden, allowing users to inspect the evolving vortex structure from arbitrary viewpoints.

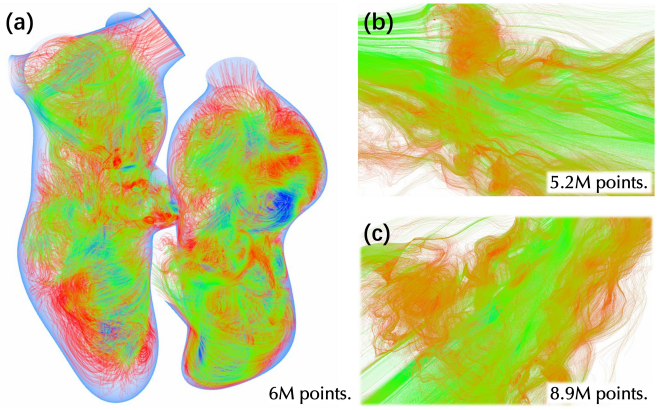


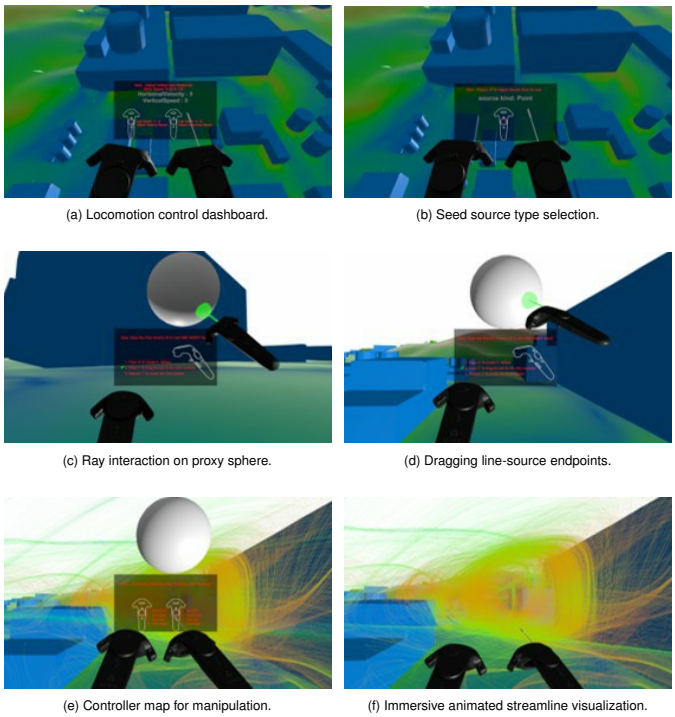
Fig. 17: Representative streamline bundles with fewer than 10^7 polyline vertices per view, illustrating that such counts are already visually very dense.

D 2D textures to Immersive 3D lines and glphys

Unlike approaches that rely on 2D textures to acquire motion maps, we directly encode and map motion properties onto 3D vertices. Compared to 3D IBFV methods, our approach avoids voxelization, thereby saving substantial computational resources. Furthermore, our geometry-based technique facilitates seamless scene integration and prevents the unnatural blending of streamlines with the background Sec. 10. For both 2D and 3D image-based methods, controlling this blending, both among the streamlines themselves and between the streamlines and the surrounding environment, remains highly challenging and often leads to severe visual clutter Fig. 19. Fundamentally, 3D IBFV still relies on 2D textures during certain processing stages, particularly when blending frames across multiple timesteps to generate trajectories. This inherent reliance makes it difficult to manage occlusion and scene integration effectively.

Regarding transparency management, it is important to clarify our design rationale. While we implemented and thoroughly evaluated explicit global depth sorting for the dense streamlines themselves, we ultimately opted against it due to the strict performance constraints of real-time rendering. For the streamtube and cell instancing visualizations presented earlier, we successfully employed multi-layer dual depth peeling, a robust Order-Independent Transparency (OIT) method. However, our empirical testing revealed that extending this exact approach to hundreds of millions of dense fine lines is counterproductive. For sub-pixel or fine line primitives, the visual improvement in depth perception provided by strict depth sorting is negligible, while the severe performance penalty fails to meet interactive real-time requirements. Current state-of-the-art OIT techniques rely on custom buffers for per-pixel sorting, yet these are generally not optimized for massive datasets either.

Interestingly, our experiments revealed a distinct advantage to employing fine lines over streamtubes: their inherently minimal screen-space footprint naturally mitigates the visual impact of incorrect occlusion sorting. As demonstrated in Fig. 20, when combined with



(a) Locomotion control dashboard.
(b) Seed source type selection.
(c) Ray interaction on proxy sphere.
(d) Dragging line-source endpoints.
(e) Controller map for manipulation.
(f) Immersive animated streamline visualization.

Fig. 18: Head-mounted views of our XR prototype: locomotion, seeding, and animated streamline visualization

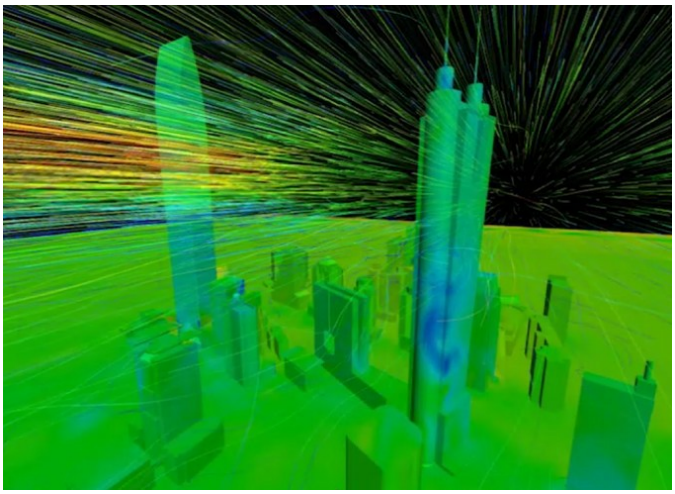


Fig. 19: Texture-based methods often fail to integrate seamlessly with the scene; trajectories are simply overlaid onto the background without any depth or transparency sorting.

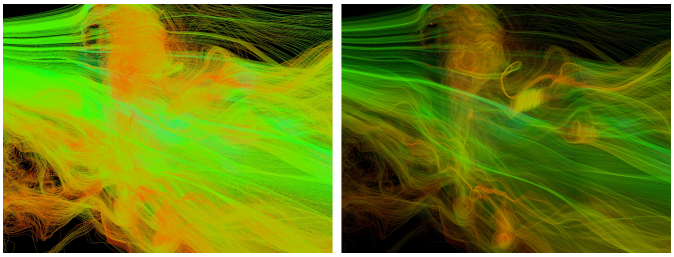


Fig. 20: For dense fine lines, the visual artifacts caused by occlusion sorting are minimal when using transparency reduction.

appropriate global transparency reduction, the depth-sorting artifacts associated with massive fine lines become practically imperceptible.

As discussed in Sec. 8, instantiating cells across 1M streamline vertices and dynamically varying their sizes reveals the underlying flow. However, this scaling effect alone is insufficient for true visual realism. Physical particles do not merely appear and shrink; they also exhibit continuous rotation driven by inertia.To achieve a realistic simulation of Lagrangian particle motion, we rely on the sequential "transfer" of attributes across Eulerian vertices. As shown in Fig. 21, we first assign a diverse set of cell meshes to the streamline vertices using a randomized function. It is important to emphasize that the actual spatial coordinates of these instantiated cells remain completely stationary. The visual illusion of motion arises as the property $\phi(x)$ evolves, transferring attributes like scale and mesh identity along the streamlines. To accurately capture particle kinematics, we introduce a time-dependent rotational component defined by $R(x) = \phi(x) + \kappa(t)$, where $\kappa(t)$ represents continuous rotation induced by inertia. This builds upon preceding time steps to yield a physically plausible depiction of transport.A notable limitation of this approach is that if the spacing between streamline vertices is too large, the lack of spatial interpolation can cause the apparent movement to manifest as abrupt flashing or jumping artifacts. Nevertheless, our method successfully demonstrates that the human visual system's perception of continuous motion can be robustly driven by the transfer of intrinsic properties, showing resilience even in the presence of spatial discontinuities.

E Precomputation for Efficiency

ParaView and several recent implementations resemble Lagrangian methods, where particles are continuously advected to produce motion. By contrast, our approach is closer to an Eulerian formulation: particles along a line remain fixed, while opacity varies with their index to create the perception of animation. We introduce a variant of Eulerian particle animation in Algorithm 1. The comparison between these two algorithms is shown in Algorithms 1 and 2, where Algorithm 1 corresponds to the ParaView method and Algorithm 2 represents our approach.

Despite improvements such as using 3D buffers to store historical information for particle lifecycles, the advection-based method is poorly suited to our needs. Storing all particle data in 3D space not only introduces significant overhead in time and memory but also complicates blending operations, much like 2D image-based approaches.

Algorithm 1 Advect Seeds

```
Require: SeedSet
  for all frame in Frames do
    for all seed in SeedSet do
      accumulate seed track
      store in buffer
    end for
    blend buffer
    render 2D image
  end for
```

Algorithm 2 Offset Opacity

```
Require: SeedSet
  for all seed in SeedSet do
    integrate by seed
    store in meshes
  end for

  for all frame in Frames do
    animated 3D lines
  end for
```

To quantify this inefficiency, we assessed rendering performance using the datasets mentioned in Sec. 9, along with an additional urban wind flow dataset (a tetrahedral mesh comprising 19,965,756 cells and 3,721,468 vertices). As demonstrated in Tab. 2, because our streamline generation pipeline operates through interactive preprocessing, our method achieves over a 7× performance improvement compared to ParaView when visualizing equivalent-scale datasets. Furthermore, through streamline segmentation, we can render streamlet animations with quantities exceeding the number of streamlines by more than two orders of magnitude, thereby significantly enhancing the visual expressiveness and dynamic representation of flow visualization.

F Rendering Features

Beyond transparency modulation, we employ high-quality shading techniques to enhance the realism and depth perception of dense streamlines. We focus primarily on illumination and occlusion, utilizing Physically

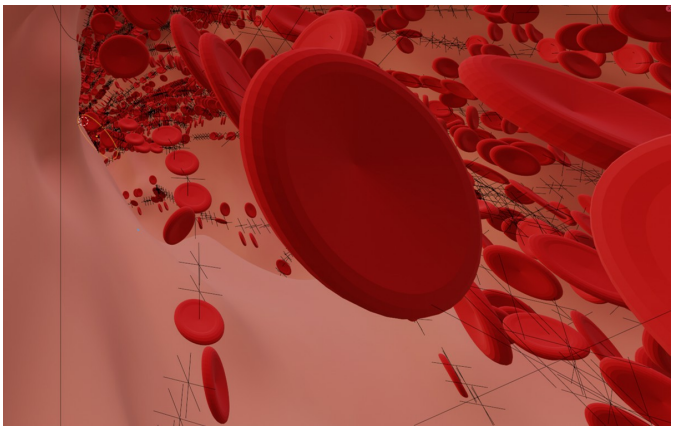


Fig. 21: Movement of cells within a flow field, noting that they strictly follow the black streamlines while rotating at a constant angular velocity.

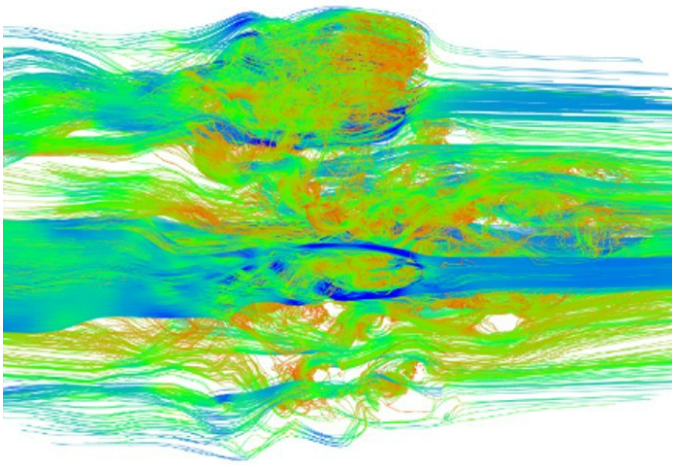
Table 2: Comparison Between ParaView & Our Method (fps)

	Power plant		Urban buildings	
	10,000	20,000	10,000	20,000
ParaView	34.9	18.2	39.9	21.7
Ours	264.2	133.2	263.9	133.7

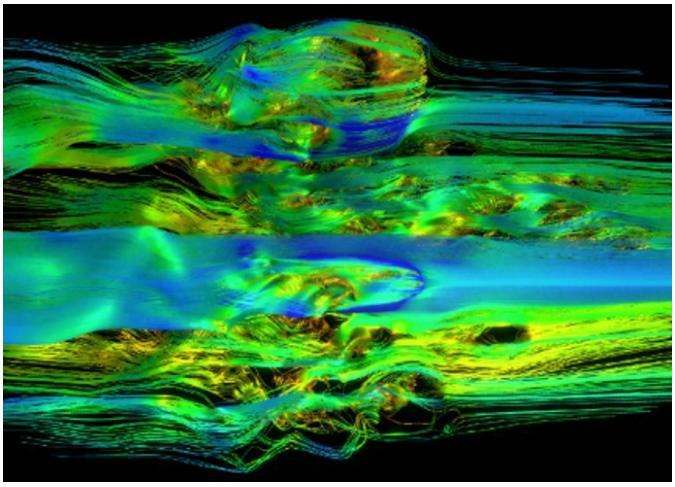
Based Rendering (PBR) and Screen Space Ambient Occlusion (SSAO), techniques prevalent in modern real-time rendering pipelines.

Notably, adapting ambient occlusion for line primitives (LineAO) necessitates real-time normal computation. To achieve this, we encode a normalized velocity vector into each streamline vertex. It is important to note that this velocity is derived from the interpolated values computed during the Runge-Kutta (RK) integration, ensuring it accurately represents the local tangent direction of the curve. By leveraging this tangent vector alongside the camera's view direction, we dynamically calculate the primitive's normal, which is essential for rendering accurate specular highlights and occlusion shadows.

While some recent AO techniques for large-scale streamtubes rely on precomputing the global spatial density distribution of the flow lines, such approaches are ill-suited for our interactive pipeline. Precomputation would introduce unacceptable latency, as the density field would need to be re-evaluated every time a user dynamically seeds new streamlines. Our real-time normal computation entirely circumvents this bottleneck. As demonstrated in Fig. 22, integrating LineAO and specular shading significantly enhances the visualization by revealing critical spatial cues, such as relative depth, spatial ordering, layering, and the complex twisting and wrapping of internal flow structures.



(a) Without LineAO and illumination.

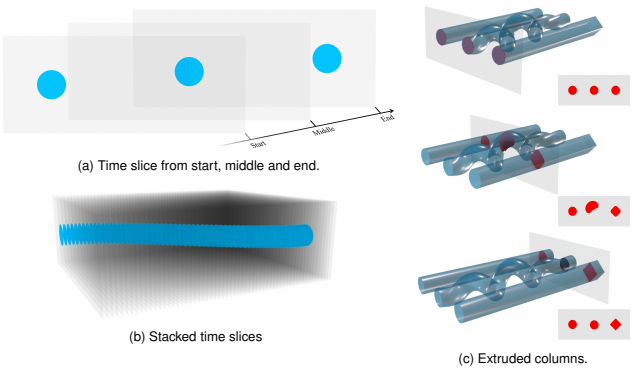


(b) With LineAO and illumination.

Fig. 22: Enhancing streamline depth perception and structural clarity using LineAO and specular shading.

G Transformation on Spacetime

To intuitively illustrate the physical advection and spacetime shear transformations discussed in Sec. 6, we provide a geometric perspective using Minkowski space. Consider a fundamental example: a 2D shape moving over time. As shown in Fig. 23a and Fig. 23b, capturing the object's discrete positions and stacking these time slices along the temporal axis extrudes the 2D shape into a continuous 3D spacetime volume. The geometry of this extrusion dictates the object's kinematics. Fig. 23c demonstrates that advancing a constant-time cutting plane through various extruded volumes yields different motion effects. Slicing a straight cylinder produces a static, persistent cross-section. Conversely, slicing a helical or oblique column yields cross-sections that continuously rotate or translate across the spatial plane. This geometric principle directly underpins our Eulerian flow maps. By extending the static 1D transparency function $\phi(x)$ into the temporal domain, we construct a 2D spacetime surface. Applying a temporal shear transformation, guided by the convection equation, slants this surface, analogous to forming the oblique columns in Fig. 23c. Consequently, as the rendering time progresses, the intersection of this sheared surface with the current time-plane naturally translates along the spatial axis. This rigorously generates the exact apparent motion of streamlets (Fig. 6b) without physically displacing the underlying streamline vertices.



(a) Time slice from start, middle and end.

(b) Stacked time slices

(c) Extruded columns.

Fig. 23: An illustration of transformation on spacetime.

H High Resolution

To achieve high-definition rendering in immersive scenes, we selected regions with large λ_2 values for automatic streamline generation. By using RK45 integration, we obtained 20 million steps (i.e., 20 million line elements in the scene), and maintained a high rendering frame rate even at 8K resolution. As shown in Fig. 24 illustrating streamline effects at various resolutions, we can observe that as resolution increases, streamlines become finer and harder to distinguish, requiring adjustments to parameters like transparency cutoff and attenuation index. Therefore, as a presentation tool, screen resolution also becomes a source of uncertainty visualization. Notice that the frame rate display in the upper right corner appears smaller as the rasterization becomes denser, indicating that as resolution increases, the frame rate noticeably declines.

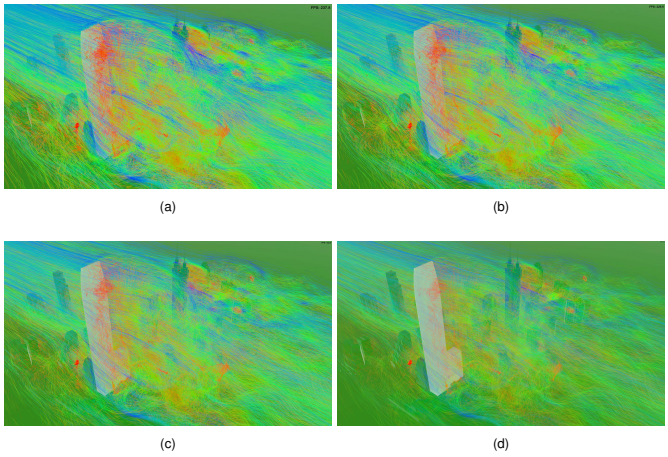


Fig. 24: The impact of resolutions on uncertainty visualization and frame rate. ***(Please enlarge the images to visualize rings and vortices,)*** and it's impressive to put on a headset and venture into the fog of animated streamlines.

I Other Virtual Reality implements for Streamlines

Our original intention was to achieve large-scale scientific visualization of fluids in a VR environment, with streamline visualization being the first consideration in our plan due to its ease of implementation. The simplest approach to placing streamlines generated by visualization software into VR involves processing the models and placing them in VR. We can export stream tubes from ParaView and attached color information to them, then imported them into Blender using the X3D format for texture rendering. Finally, we exported the models and textures for placement in Unity, where we applied some sophisticated post-processing techniques to showcase stunning static streamlines. We also explored some ParaView plugins designed to render objects directly from ParaView to Unity's viewport, but found their effectiveness to be lacking.

It's important to note that there is no perfect solution using off-the-shelf state-of-the-art technology. We investigated and verified others methods, two of which stood out: first, ParaView's XR plugin, and second, Omniverse's integration with OpenUSD. These methods achieve impressive stereoscopic visual effects, but they are still just visual effects. Their use of VR remains primarily focused on post-processing flow fields for desktop viewing, with VR goggles used as an additional viewing tool.

J Versatile and Efficient Implementations

This section details our supplementary materials, where we adapt our algorithm into multiple practical implementations to demonstrate its efficiency and versatility.

Interactive Webpage: First, we highly recommend trying the interactive WebXR demos. These allow you to adjust animation parameters

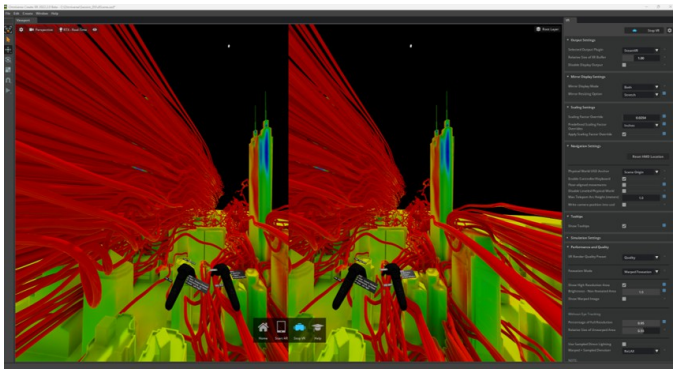


Fig. 25: Using "Create XR" to establish a connection to the shared Nuclear session for scientific visualization of streamlines and scenes.

on the fly, highlighting our two main contributions: rendering efficiency and visual clarity. The algorithm is so efficient that dense animated streamlines can run smoothly even on mobile browsers. (Please be careful with the third dataset—the urban wind field—as its massive scale may cause lag on smartphones). By tweaking the parameters, you can effectively reduce visual clutter and create rich dynamic effects. As our preset views demonstrate, parameter tuning should depend not just on the flow field, but also on the observer's specific viewpoint. An on-screen FPS counter is also included so you can easily verify performance across different devices.

The web demos feature three datasets: cardiac turbulence, liver perfusion, and urban wind, Show in Fig. 26. The first two are lightweight and run easily on most devices. For the urban wind dataset, we strongly recommend using a high-resolution display or an XR headset to fully experience the immersive optical flow and spot tiny vortex structures.

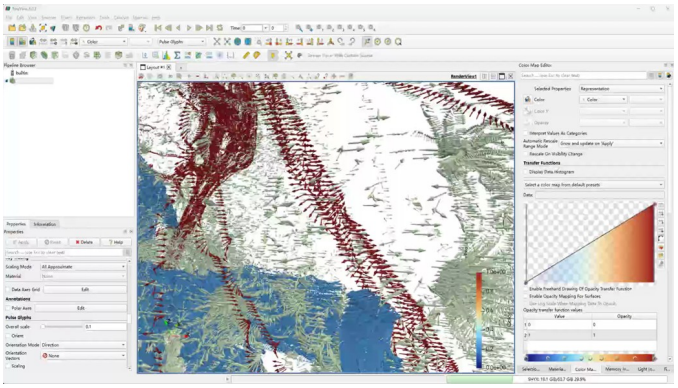
Open-Source ParaView Plugins: To integrate our method into standard scientific workflows, we provide two open-source plugins for ParaView 6.0.x: the *Animated Streamline Representation* and the *Pulse Glyph Representation*. Both plugins share the same underlying Eulerian motion mapping theory (as discussed in Sec. 8) and rely on an "Animation Coordinate Array" (typically the integration time, $s(\mathbf{x})$) to drive the spatiotemporal phase.

Because they share the same core mathematical model, they feature a set of identical control parameters to shape the animation:

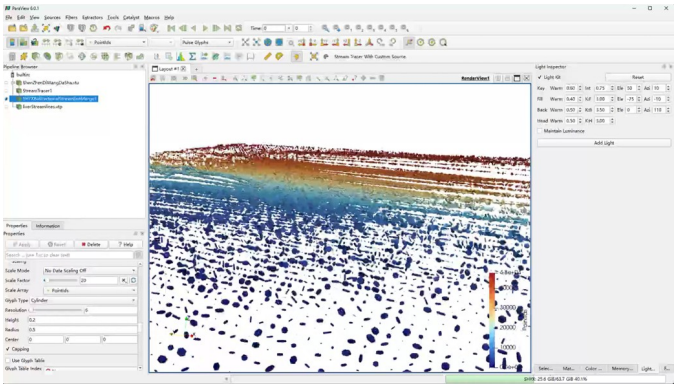
- **Integration Scale**: Controls the spatial frequency of the streamlets. A higher value results in shorter, more densely packed animated segments along the line.
- **Time Scale**: Determines the speed of the apparent motion (γ).
- **Trunc**: Acts as a truncation factor for the tail of the streamlet, defining how abruptly the trailing effect cuts off.
- **Pow**: Controls the curvature of the decay profile (λ). A value of 1.0 yields a linear fade, while higher values create sharper, exponential-like decays.

Animated Streamlines: This plugin operates entirely on the GPU. It intercepts ParaView's rendering pipeline and injects custom GLSL shader replacements to dynamically modulate the fragment opacity. Aside from the shared parameters, it includes an *Opacity Scale* parameter, which acts as a global multiplier to adjust the overall transparency of the dense line clusters, effectively reducing visual clutter without altering the motion frequency.

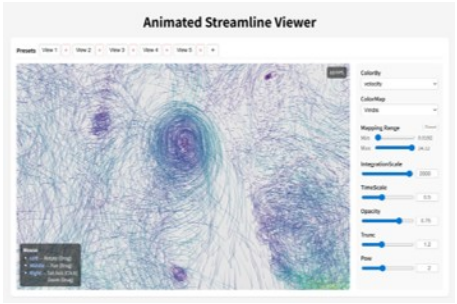
Pulse Glyphs: Instead of rendering lines, this plugin instantiates geometry (e.g., spheres, arrows, or prisms) at the streamline vertices. To create the illusion of flow, the spatiotemporal phase is mapped to the scale and rotation of the glyphs using CPU-side multithreading (*vtkSMPTools*). When using this plugin, the initial animation might appear chaotic. To achieve a realistic flow effect, you should first reduce the *Pulse Overall Scale* to shrink the maximum size of the instanced geometry. Then, adjust the shared *Integration Scale* so that the size of the glyphs shrinks progressively along the streamline, mimicking



(a) Apparent motion induced by the collective rotation of massive arrow glyphs.

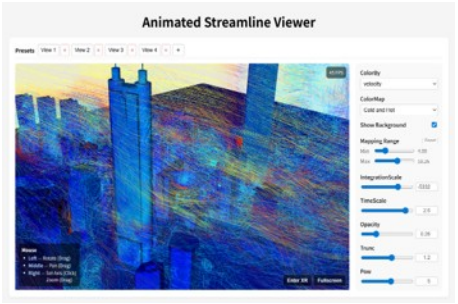


(b) Real-time Visualization of Massive Particles with Irregular Motion along Streamlines in ParaView.



(c) Intracardiac Turbulence

(d) Hepatic Perfusion



(e) Airflow around Building Complexes.

Fig. 26: Web-based Interface and ParaView Plugin for Efficient Animated Streamlines.

the fading tail of the streamline decay profile. You can also enable *Pulse Affects Rotation* and adjust the *Rotation Sweep* axes to make the particles spin dynamically as they move, shown in.

To make the particle motion look more natural and less uniform, check the *Shuffle* option. As shown in Fig. 26b, this assigns pseudo-random sizes and orientations to the glyphs, creating a chaotic, particle-laden flow effect. However, to keep the animation stable, the random seed for each glyph must stay consistent across frames. Because fluctuating CPU frame rates make it extremely difficult to perfectly synchronize the real elapsed time with the random number generator, our plugin intelligently switches to using the *Render Frame count* rather than elapsed time to drive the shuffle animation, ensuring jitter-free visual stability.

Standalone Applications: Finally, our teaser video demonstrates both a Meta Quest app and a desktop app. If you try the Quest application, please be mindful of the headset's hardware limits: only enable the "Show as Particles" (animated glyphs) option when rendering a small number of streamlines, otherwise the heavy geometry load may cause severe lag and XR sickness. As for the desktop application, we currently provide a non-XR version purely for performance benchmarking. Due to current hardware and software constraints regarding mixed-reality passthrough, we do not yet offer a desktop VR version, but we plan to release one in the future.